AD NUMBER

AD911484

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution authorized to U.S. Gov't. agencies only; Test and Evaluation; MAR 1973. Other requests shall be referred to Air Force Avionics Lab., Attn: AAM, Wright-Patterson AFB, OH 45433.

AUTHORITY

Air Force Avionics Lab ltr dtd 24 Feb 1978

AFAL-TR-73-114

AEROSPACE MULTIPROCESSOR
FINAL REPORT

Robert L. Davis
Sandra Zucker, et al

Burroughs Corporation
Defense, Space and Special Systems Group
Advanced Development Organization
Paoli, Pennsylvania 19301

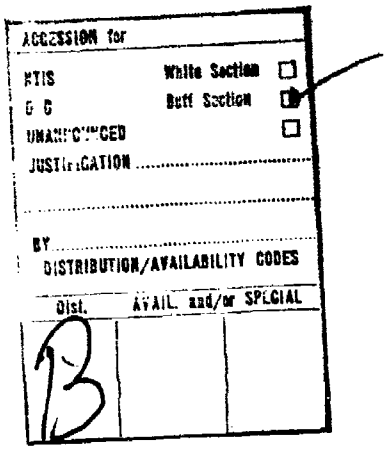
TECHNICAL REPORT AFAL-TR-73-114

June 1973

Air Force Avionics Laboratory
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.



Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AEROSPACE MULTIPROCESSOR
FINAL REPORT

Robert L. Davis
Sandra Zucker, et al

FOREWORD

This Final Engineering Report was prepared by the Burroughs Corporation, Defense, Space and Special Systems Group, Advanced Development Organization, Paoli, Pennsylvania. The work was accomplished under USAF Project 6090 entitled "Avionics Data Handling Technology", Task 01 entitled "Avionics Information Processing" and Contract No. F33615-70-C-1773 entitled "Aerospace Multiprocessor." The work was administered under the direction of Mr. D. Brewer, Air Force Avionics Laboratory, AFAL/AAM, Wright-Patterson AFB, Ohio.

This report covers work conducted from June 1970 to March 1973 and was submitted by the authors March 1973.

The authors, Mr. Robert Davis and Mrs. Sandra Zucker, are grateful for the help and contributions of many of their associates in the Advanced Development Organization for the documentation, wiring, machining, board layout and fabrication, and artwork generation necessary to build the multiprocessor and for the help of their associates in Advanced Development and Technical Publications in the writing, drafting, typing, and proofreading necessary to produce this report. The authors are especially grateful to Messrs. Peter Molloy and Gilbert Reid for their help in the fabrication and debugging of the multiprocessor; Messrs. Melvin Brooks and Carl Campbell for their help in writing and debugging the control and demonstration programs; Messrs. Ulbe Faber and Richard Bradley for the design of the Switch Interlock; and Mr. John T. Lynch, Director of Advanced Development and Messrs. Dewey Brewer and Ralph Barrera of the Avionics Laboratory (AFAL/AAM) for their patience and support throughout this program.

This technical report has been reviewed and is approved for publication.

COZIER S. KLINE
Colonel, USAF
Chief
System Avionics Division

ABSTRACT

The aerospace multiprocessor described is based upon a modular, building block approach. An exchange concept that is expandable with the number of processors, memory modules, and device ports, was developed whose path width is a function of the amount of serialization desired in the transmission of data and address through the exchange. The processors (called Interpreters) are microprogrammable utilizing a 2-level microprogram memory structure and were designed for implementation with large scale integrated circuits. The modularity exhibited in the Interpreters is in the size of the microprogram memories and in the word length of the Interpreters from 8 bits through 64 bits in 8-bit increments.

The specific implementation of the exchange for the aerospace multiprocessor is for five processors, eight memory modules, and eight device ports with eight wires each carrying four serial bits of data through the exchange. The processors each have word lengths of 32 bits with a 512 word × 15 bit first-level microprogram memory and a 256 word × 54 bit second-level microprogram memory.

A simplified control program based upon concepts for a modular executive structure, and some user type programs were written for demonstration of the aerospace multiprocessor.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Cont'd)

SECTION I

INTRODUCTION

This final report describes the results of work performed by the Advanced Development Organization of Burroughs Defense, Space and Special Systems Group for the Air Force Avionics Laboratory, Wright-Patterson Air Force Base under contract F33615-70-C-1773. The purpose of this program was to fabricate an aerospace multiprocessor utilizing large scale integrated circuits with techniques developed under contract F33615-69-C-1200 by Burroughs for the Avionics Laboratory.

The aerospace multiprocessor is made up of five identical microprogrammable, LSI processors called Interpreters connected to devices and memory modules by an exchange called a Switch Interlock. Since the intent of the contract was to produce only those parts of a multiprocessing system (processors and exchange as shown in Figure 1) not readily available in "miniaturized" form, the system is completed with commercially available memory modules, power supplies, and devices as shown in Figure 2. In this figure, the items delivered are shown within the dotted line. The Switch Interlock module comprises the "network" shown by the connected lines on the bottom half of Figure 2. The system characteristics for the aerospace multiprocessor are listed in Table I.

The remainder of this report consists of seven sections and seven appendices. Section II describes the LSI, microprogrammable processor (called an Interpreter), consisting of three types of logic parts utilizing discretionary-wired LSI arrays, two types of microprogram memories and a loader for loading these two memories. Also included is a discussion of the rationale for splitting the microprogram memory into two parts, based on work done by Mr. Ernest Trimbur.

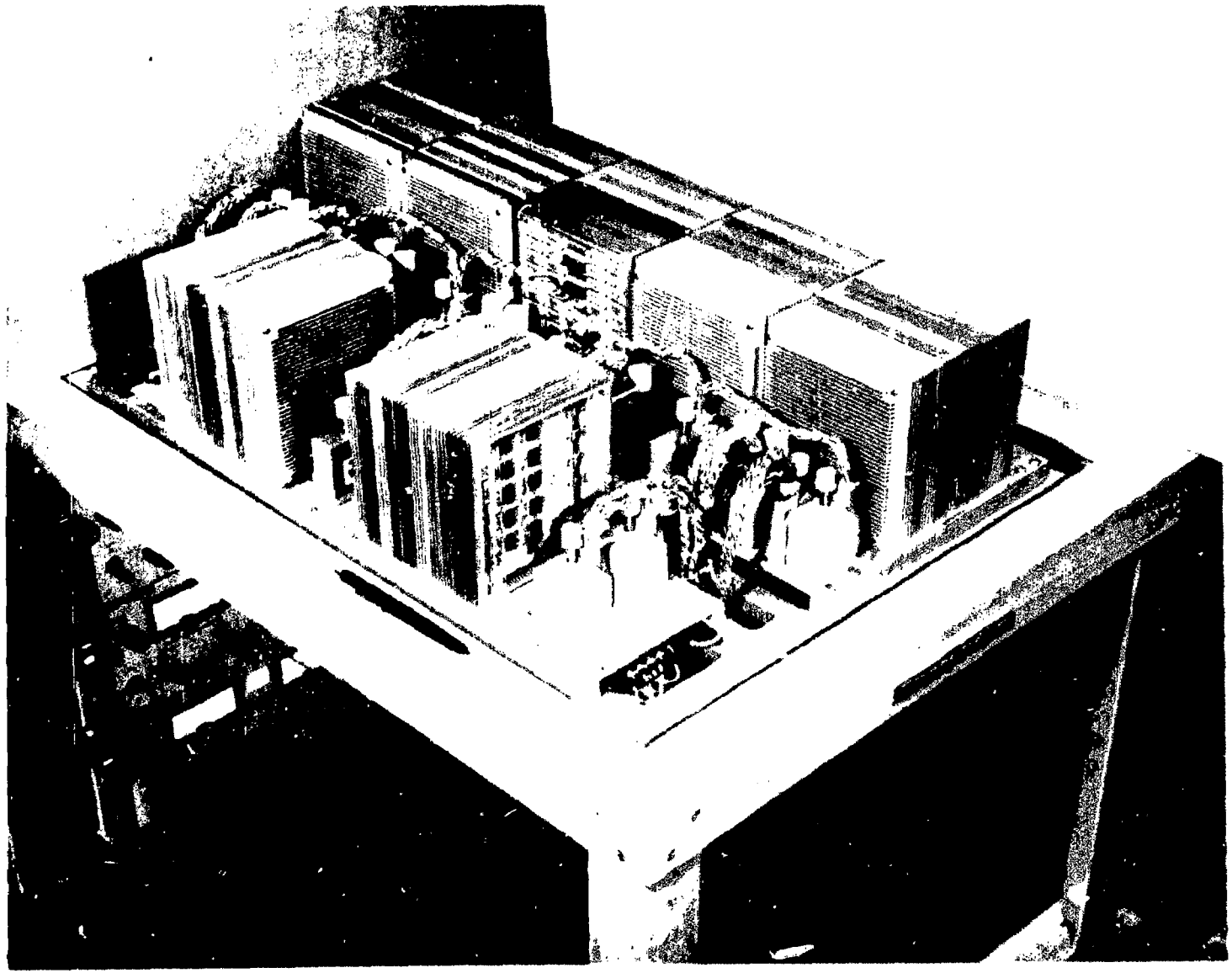


Figure 1. Basic Aerospace Multiprocessor

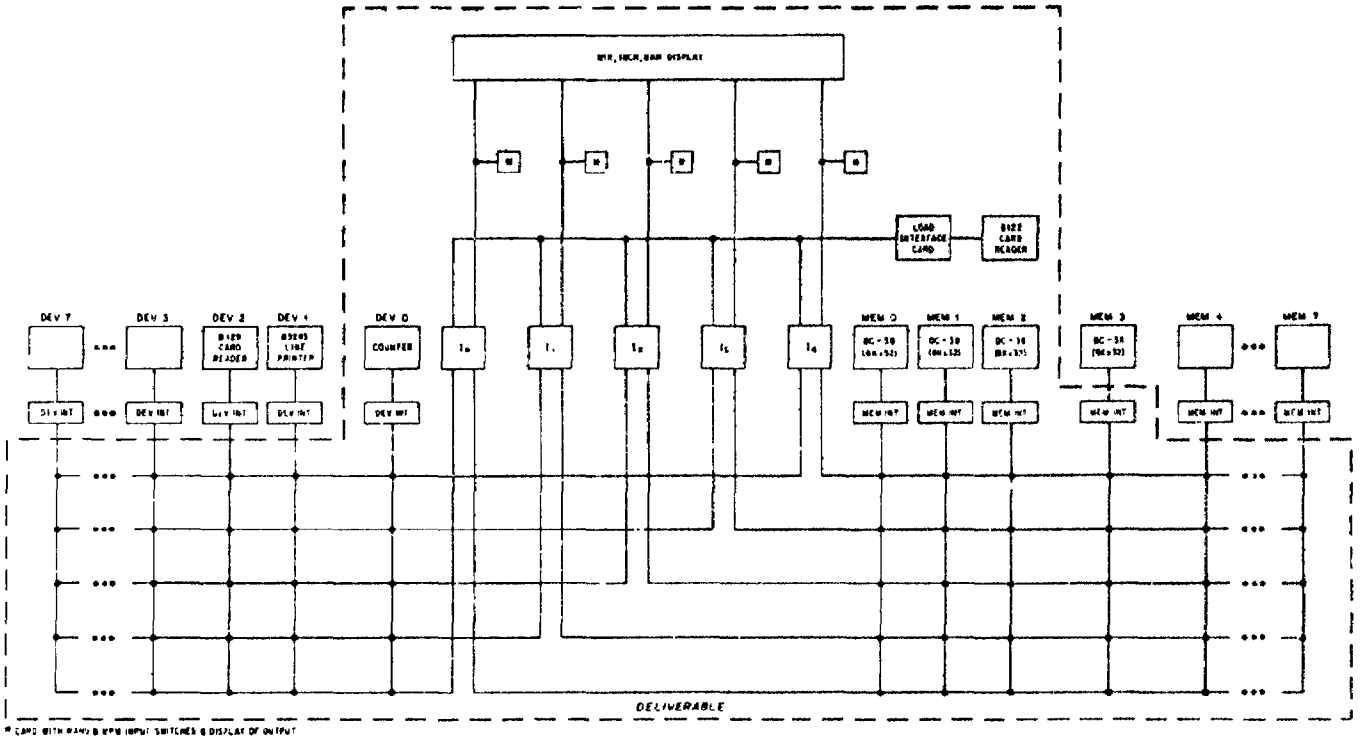


Figure 2. LSI Multi-Interpreter System Block Diagram

Table I. Aerospace Multiprocessor, System Characteristics Summary

5 Interpreters

32-bit word length

2.5 mHz clock rate

Discretionary Routed TTL, LSI

512 words (expandable to 1024 words) by 15 bits, read/write MPM

256 words by 54 bits, read/write Nanomemory

Volume: 5.75 in. × 5.1 in. × 6 in. without connectors.
5.75 in. × 5.1 in. × 10 in. with connectors

Typ. Power: 42 watts for LSI arrays
4 watts for loader
44 watts for MPM and Nanomemory
} at +5 volts dc

3 Memory Modules

Datacraft DC-38

3-wire, 3D, coincident current core

Read/write, random access

8K words (expandable to 16K words) by 32 bits per module

350 ns access/900 ns cycle

Volume: 19 in. × 19 in. × 5 1/4 in.

Typ. Power: 6A at 117 Vac

1 Switch Interlock

5 Interpreter ports

Serial data interface of 8 wires of 4 serial bits each

8 serial interfaces for memory modules (32 bits wide)

8 serial interfaces for device ports (32 bits wide)

Volume: 5.75 in. × 5.1 in. × 22 in. with connectors

Typ. Power: 72 watts at +5 volts dc

Section III includes a general discussion of multiprocessor interconnection and a description of the hardware specifically needed for multiprocessing. This hardware includes the exchange for interconnecting processors to memories and devices, clock and power control, a "real-time" clock, a time-out counter, and the hardware necessary for one Interpreter to lock other Interpreters out of selected tables in memory. Also included in this section is a description of the system power distribution.

Section IV describes the packaging of the multiprocessor for its laboratory environment and briefly discusses the LSI partitioning and possible future implementations.

Section V is a detailed discussion of the Interpreter operation as a single processor, centering primarily on the fetching, execution, and sequencing of microprogram instructions and the condition testing involved in the microprogram instruction's successor determination.

Section VI is a detailed discussion of the Switch Interlock operation. The conflict resolution problem in accessing memories and "locking" to devices is discussed along with the handshaking between the Interpreters and the Switch Interlock in performing memory and device operations. Detailed timing diagrams are given for all Switch Interlock operations.

Section VII describes the microprogramming of the Interpreter and gives the syntax and semantics and examples for all Interpreter operations.

Section VIII is divided into two parts. The first part describes the simplified control program used to control the multiprocessor with its associated task tables in memory and also describes the method for loading either tasks or the control program into the Interpreter's microprogram memories from "S" memory. The second part of this section describes the six programs written to be executed as user tasks in the demonstration of the multiprocessor. This section is concluded with a short discussion of the confidence routines that were used during debugging of the Interpreters and which could be modified to run under the operating system for on-line confidence checks of the Interpreters.

Appendix I is a historical review of microprogramming written by Dr. Earl Reigel. Appendix II is a copy of the final report from Texas Instruments, Inc. on the discretionary-wired LSI used in the Interpreters. Appendices III-VI are details for the use of TRANSLANG, an assembler for Interpreter microprograms. Appendix VII is a glossary.

SECTION II

INTERPRETER HARDWARE BUILDING BLOCKS

The Interpreter is composed of four logic package types: the Logic Unit (LU), the Control Unit (CU), the Memory Control Unit (MCU), and the Loader (LDR). The microprograms which provide the control functions are contained in two memories: the Microprogram Memory (MPM) and the Nano program Memory (Nano or NM). These units and their interconnections are shown in Figure 3.

The unique split memory scheme for microprogram memories allows a significant reduction in the number of bits for the microinstruction storage. It should be noted, however, that a single microprogram memory scheme (MPM and Nano combined) could also have been used, potentially increasing the clock rate of the system. In addition, the cycle rates of the memories could be altered, to gain speed or reduce cost, without any redesign of the logic packages. In fact, a variety of memory organizations (single memory and different split memory configurations) and memory speeds have been implemented in other Interpreter based systems, thus providing a range of cost/speed trade-offs.

The LU performs the required shifting, arithmetic, and logic functions as well as providing a set of scratch pad registers and data interfaces to and from the Switch Interlock (SWI). Of primary importance is the modularity of the LU, providing expansion of the word length in 8-bits increments from 8 bits through 64 bits using the same functional unit. The word length of the Interpreters used in the aerospace multiprocessor is 32-bits.

The CU contains a condition register, logic for testing the conditions, a shift amount register for controlling shift operations in the LU, and part of the control register used for storage of some of the control signals to be sent to the LU.

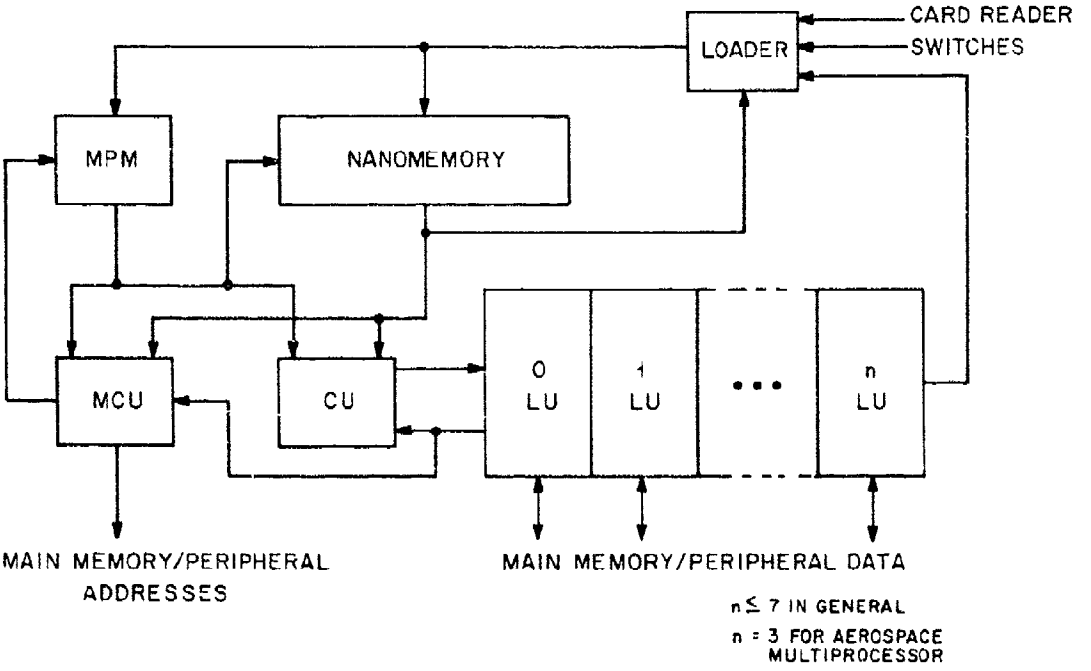


Figure 3. Interpreter Block Diagram

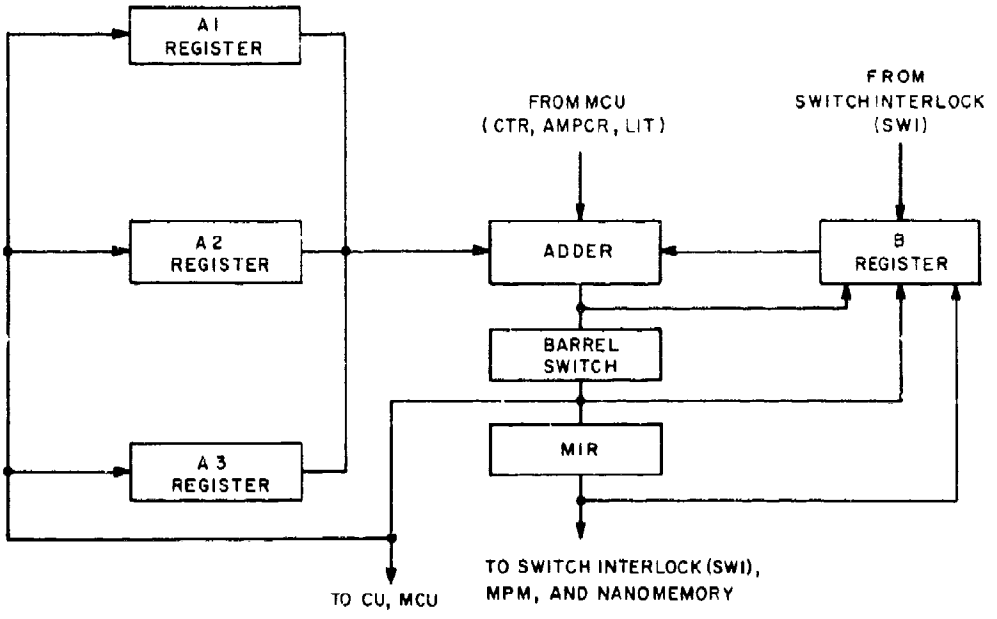


Figure 4. Logic Unit Block Diagram

The MCU provides addressing logic to the Switch Interlock for data accesses, controls for the selection of microinstructions, literal storage, and counter operation. This unit is also expandable when larger addressing capability is required. The Loader (LDR) enables the MPM and Nanomemory to be loaded from either switches, a card reader, or programmatically from the LU.

LOGIC UNIT (LU)

A functional block diagram of the LU is shown in Figure 4. The design of the LU is predicated upon implementation with one LSI silicon slice per eight bits. The present 8-bit LU is implemented with two LSI slices.

Registers A1, A2, and A3 are functionally identical. Each temporarily stores data and serves as a primary input to the adder. Selection gates permit the contents of any A register to be used as one of the inputs to the adder. Any of the A registers can be loaded with the output of the barrel switch.

The B register is the input buffer (from the Switch Interlock). It serves as the second input to the adder and can also collect certain side effects or arithmetic operations. The B register may be loaded with any of the following (one per instruction):

1. The barrel switch output
2. The adder output
3. The data from the Switch Interlock
4. The MIR output
5. The carry complements (from the adder) of 4- or 8-bit groups with selected zeros (for use in decimal arithmetic or character processing)
6. The barrel switch output ORed with the adder output
7. The barrel switch output ORed with the data from the Switch Interlock
8. The MIR output ORed with 1, 2, 5, or 6 above.

The output of the B register has true/complement selection gates which are controlled in three separate sections: the most significant bit, the least significant bit, and all the remaining central bits. Each of these parts is controlled independently and may be either all zeros, all ones, the true contents or the complement (ones complement) of the contents of the respective bits of the B register. The operation of these selection gates affects only the output of the B register. The contents remain unchanged.

The MIR primarily buffers information being written to main system memory or to a peripheral device. It is loaded from the barrel switch output and its output may be sent to the Switch Interlock, to the B register, or to the data input of the MPM or Nanomemory for programmatic loading.

The adder in the LU is a modified version of a straightforward carry look-ahead adder such as that discussed by MacSorley[1] and others. Therefore, the details of its operation will not be included.

Inputs to the adder are from selection gates which allow various combinations of the A, B, and Z inputs. The A input is from the A register output selection gates and the B input from the B register true/complement selection gates. The Z input is an external input to the LU and can be:

1. The 8-bit output of the counter of the MCU into the most significant 8 bits with all other bits being zeros.

2. The 8-bit output of the literal register of the MCU into the least significant 8 bits with all other bits being zeros.

3. The 12-bit output of the alternate microprogram count register (AMPCR) right justified into the middle 16 bits and the (wired) Interpreter number right justified in the remaining four bits of the middle 16 bits. All other bits are zeros.

4. All zeros.

Using various combinations of inputs to the selection gates, any two of the three inputs can be added together, or can be added together with an additional "one" added to the least significant bit. Also, all binary Boolean operations between the A and B and between the B and Z adder inputs and most of the binary Boolean operations between the A and Z adder inputs can be done.

The barrel switch is a matrix of gates that shifts a parallel input data word any number of places to the left or right, either end-off or end-around, in one clock time.

The output of the barrel switch is sent to:

1. The A registers (A1, A2, A3)

2. The B register

3. Memory Information Register (MIR)

4. Least significant 16 bits to MCU (registers BR1, BR2, MAR, AMPCR, LIT, CTR)

5. Least significant 5 bits to shift amount register (SAR) in the CU.

CONTROL UNIT (CU)

One CU is required for each Interpreter. The design of the CU is predicated upon implementation with one LSI silicon slice, but is presently constructed with two LSI slices. Major sections of this unit (Figure 5) are: the shift amount register (SAR), the condition register, part of the control register (CR), the MPM content decoding, and part of the clock control.

The functions of the SAR and its associated logic are:

1. To load shift amounts into the SAR to be used in the shifting operations. Left end-off shifts require a shift amount equal to the "word length complement" of the number of positions to be shifted. ("Word length complement" is defined as the amount that will restore the bits of a word to their original position after an end-around shift of N followed by an end-around of the "complement" of N. For the 32-bit word length in the aerospace multiprocessor, this is the 2's complement.)

2. To generate the required controls for the barrel switch shift operation indicated by the controls from the Nanomemory.

3. To generate the "word length complement" of the SAR contents and load this value back into the SAR.

The condition register section of the CU performs four major functions:

1. Stores 12 resettable condition bits in the condition registers. The 12 bits of the condition register are used as error indicators, interrupts, status indicators, and lockout indicators.

2. Selects 1 of 16 condition bits (12 from the register and 4 generated during the present clock time in the Logic Unit) for use in performing conditional operations.

3. Decodes bits from the Nanomemory for resetting, setting, or requesting the setting of certain bits in the condition register.

4. Resolves priority between Interpreters in the setting of global condition (GC) bits.

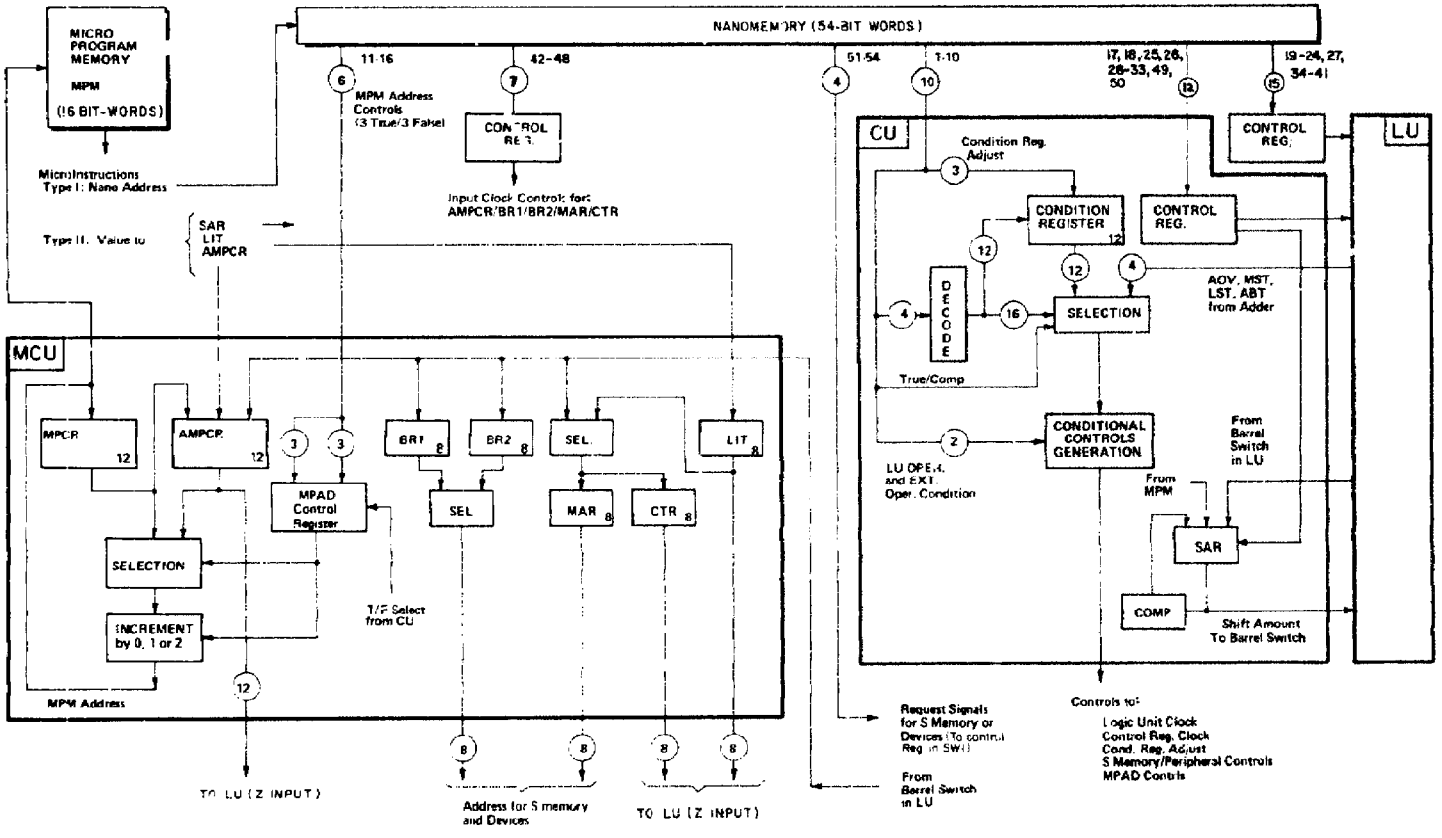


Figure 5. Interpreter Functional Units

The control register is a register that stores 38 of the 54 control signals from the Nanomemory that are used in the LU, CU, and MCU for controlling the execution phase of a microinstruction. Twelve of the 38 outputs from the Nanomemory are stored in the CU. Four of the other 38 Nanomemory outputs are controls to the Switch Interlock and are stored there. The other 22 of the 38 Nanomemory outputs are stored in a part of the control register physically located in the Nanomemory.

The MPM content decoding determines (based upon the first four bits of the MPM) whether the MPM output is to be used as a Type I instruction (Nanomemory address) or as a Type II instruction (literal). Several decoding options are available. The particular option chosen is described in the Interpreter Microprogramming section of this report.

MEMORY CONTROL UNIT (MCU)

One MCU is required for an Interpreter in the aerospace multiprocessor, but a second MCU could have been added to provide additional memory addressing capability. The design of the MCU is predicated upon implementation with one LSI silicon slice, but is presently constructed with two LSI slices. This unit has three major sections (Figure 5):

1. The microprogram address section contains the microprogram count register (MPCR), the alternate microprogram count register (AMPCR), the incrementer, the microprogram address control register, and associated control logic. The output of the incrementer addresses the MPM for the sequencing of the microinstructions. The AMPCR contents are also used as one of the Z inputs to the adder in the LU.

2. The memory/device address section contains the memory address register (MAR), base registers one and two (BR1, BR2), the base register output selection gates, and the associated control logic.

3. The Z register section contains registers which are two of the Z inputs to the LU adder: a loadable counter (CTR), the literal register (LIT), selection gates for the input to the memory address register and the loadable counter and their associated control logic.

NANOMEMORY (N MEMORY)

The Interpreter is controlled by the output of the 54-bit wide Nanomemory which may be implemented with a read/write memory, a read-only memory, wired logic, or a combination of the three. The present implementation is a 256-word by 54-bit read/write semiconductor random access memory using the Fairchild 93410, a 256-word by 1-bit package.

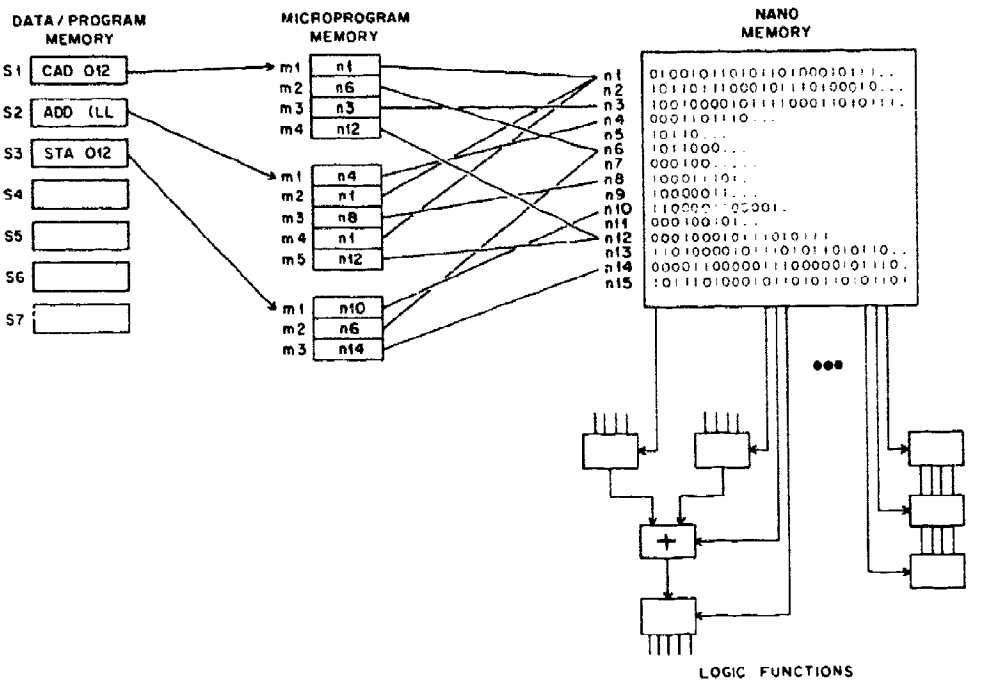


Figure 6. Instruction Memory Hierarchy

Each of the 54 bits represents a unique enable line for the gates and flip-flops within the LU, the CU, and the MCU. Each Nanomemory word represents a microinstruction that is executed by the simultaneous presentation of a specific enable pattern for the 54 outputs, represented by corresponding ones and zeros in its word. The definition of these bits is presented in the microprogramming section.

A unique feature of the Interpreter-Based System with its separate Nanomemory and Microprogram Memory (Figure 5) is that the explicit enable lines for each microinstruction need be stored in the Nanomemory only once (regardless of the number of times that a specific microinstruction is needed in a program). To accomplish this saving in memory, the Microprogram Memory (MPM) contains the address in the Nanomemory where the explicit ones and zeros are stored that are needed to execute that instruction' type rather than the full micro-instruction. Thus, several microprogram sequences which use the same micro-instruction (e.g., transfer A to B) need only store in the Microprogram Memory the address of the Nanomemory word containing that microinstruction. Figure 6 illustrates this feature.

MICROPROGRAM MEMORY (MPM)

Each Interpreter requires a source of microprogram instructions to define the operation of the Interpreter.

Two possible solutions for providing this source of microprogram instructions are listed below:

1. A semiconductor MPM. This memory can be a read-only memory (ROM) if the Interpreter is to be dedicated to the function defined by the ROM. A read-write memory can be used for experimental purposes or when the function of the Interpreter might be changed, such as reconfiguration in a multiple Interpreter system. In this instance, the system could afford to wait while the MPM was reloaded from a remote microprogram store accessed via the Switch Interlock.

2. A buffer into a slower-speed, wider-word memory.

In presently deliverable large scale integration form of the Interpreter, the MPM is also implemented with Fairchild 256-word by 1-bit bipolar, nondestructive readout semiconductor memory packages. Both the MPM and the Nanomemory can be loaded from an external loader, switches or programmatically from its own MIR. The basic MPM is expandable in blocks of 256 words, and can be expanded up to 1024 words in the present Interpreters.

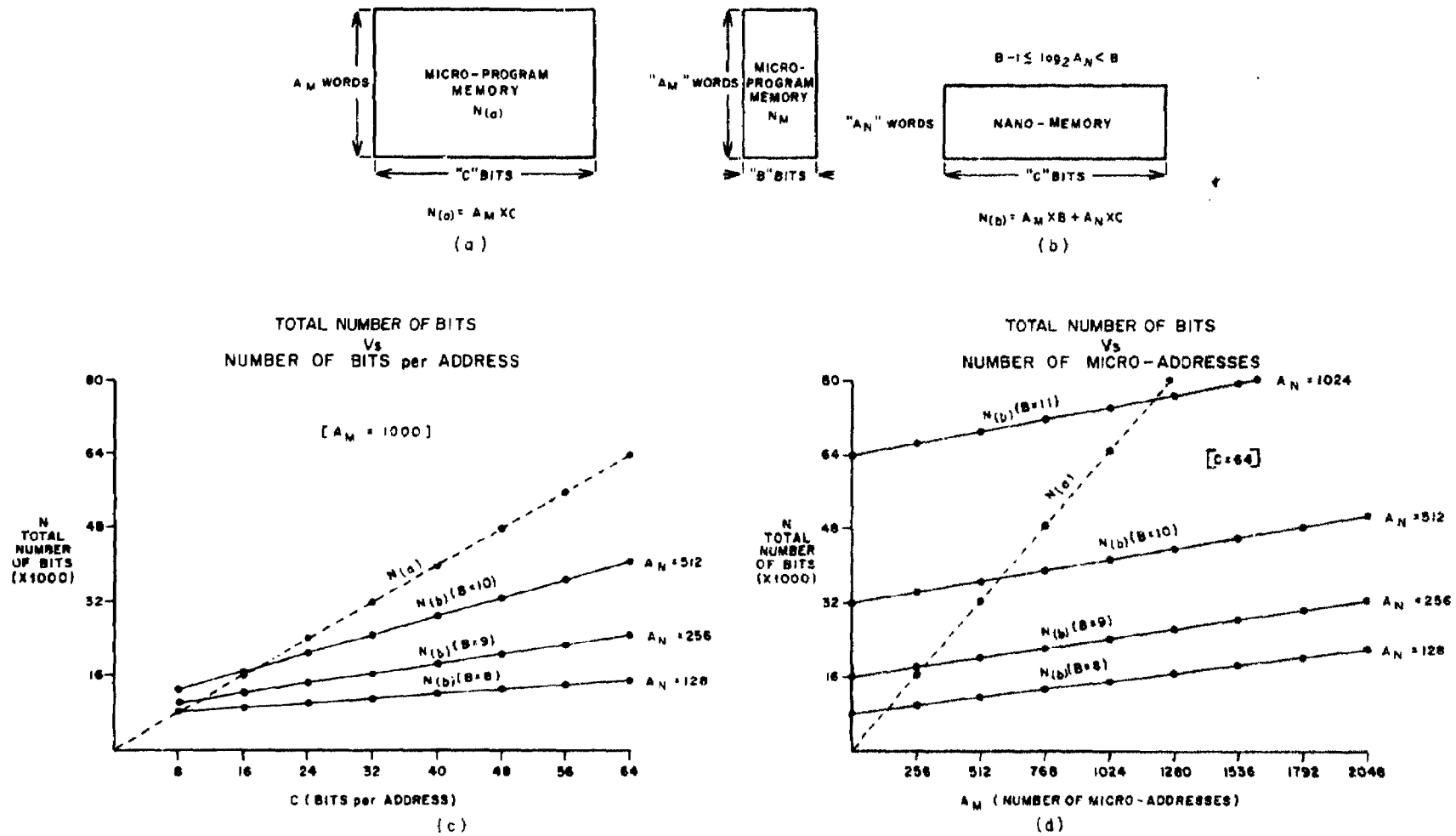


Figure 7. One Memory vs. Two Memory Implementation

Microprogram Memory Considerations

The potential advantage of dividing what is considered to be the Microprogram memory into two parts is more graphically illustrated by comparing the total memory requirements of the two approaches shown in Figure 7.

The total number of bits ($N_{(a)}$) in Figure 7 (a) is given by $N_1 = A_M \times C$. The total number of bits ($N_{(b)}$) in Figure 7 (b) is given by $(A_M \times B) + A_N \times C$. A plot of the total number of bits vs. B and C and a plot of the total number of bits vs. A_M and B for both approaches are shown in Figures 7 (c) and (d).

From these figures, it is obvious that as A_N approaches A_M, one memory is the proper approach. Two factors affect the relationship between A_M and A_N. One is that literal values (type II instructions) used for shift amounts, jump addresses and 8-bit literals, that appear in the Microprogram memory, make no reference to the Nanomemory. Second, repetitive use of the same nanoinstruction causes an increase in A_M without adding words to the Nanomemory. Some sample program statistics are shown in Figure 8. This figure shows, for four sample programs, the total number of microprogram and nanomemory words, the total number of bits for both the one and two memory approaches and the percentage and actual value of the number of bits saved using the two instead of the one memory approach. In addition, this table shows the comparison among the number of literals (type II instructions), the number of Nanomemory references (type I instructions), and the number of Nano memory locations in the four sample programs.

It should be remembered that the two memory approach would require memories with approximately twice as fast an access time (and hence are more expensive per bit) because both memories must be accessed sequentially within one clock time.

Memory cost per bit vs. memory cycle time is shown in Figure 9, where the vertical bars indicate the range on these prices which were gathered during January, 1972. Although the absolute prices have decreased, the relative pricing should still be valid. Several cost factors (C. F.'s) are shown for memory speeds having a 2:1 ratio. The cost factors are simply the ratio of the price of the faster memory to that for the slower memory. The higher cost factor encountered when crossing technology boundaries should be noted.

The solid lines in Figure 10 show the actual cost savings of the two memory approach for the four sample programs taking into account the difference in memory prices for the two approaches.

Also it is important to realize that many applications require a writable Microprogram memory. This means that the entire memory in the one memory approach must be read-write, while with the two memory approach, the Nanomemory could be read-only with the Microprogram memory being read-write. (In fact the Nanomemory could even be partly read-only and partly read-write.) This is shown by the dashed lines

	PROGRAM STATISTICS					TOTAL BITS		2 MEMORY TECHNIQUE BIT BREAK DOWN				
TASK	A_M			A_N	Rov.	$N_{(a)}$	$N_{(b)}$	N_M	N_N	BIT SAVINGS		
	TOTAL	TYPE I (%)	TYPE II (%)							$N_{(a)}-N_{(b)}$	ΔN_I	ΔN_{II}
D-825 EMULATION	3337	2224 (67%)	1113 (33%)	964	2.31	187K	108K	54K	54K	79K (42%)	35K	44K
B-300 EMULATION	3265	1996 (61%)	1269 (39%)	624	3.20	183K	87K	52K	35K	96K (52%)	45K	51K
DISK CONTROLLER	1288	910 (71%)	378 (29%)	244	3.73	72K	34K	21K	14K	38K (52%)	23K	15K
LANGLAGE DESIGN SYSTEM	659	394 (60%)	265 (40%)	244	1.61	37K	25K	11K	14K	12K (32%)	2K	10K

Rov. = AVERAGE REPETITIVE COMMAND FACTOR = TYPE I / A_N

ΔN_I = BIT SAVING DUE TO NON-REPETITIVE STORAGE = $I \times C - [A_N \times C + I \times B]$

ΔN_{II} = BIT SAVING DUE TO SHORTENED TYPE II WORD LENGTH = $II \times [C-B]$

Figure 8. Sample Program Statistics

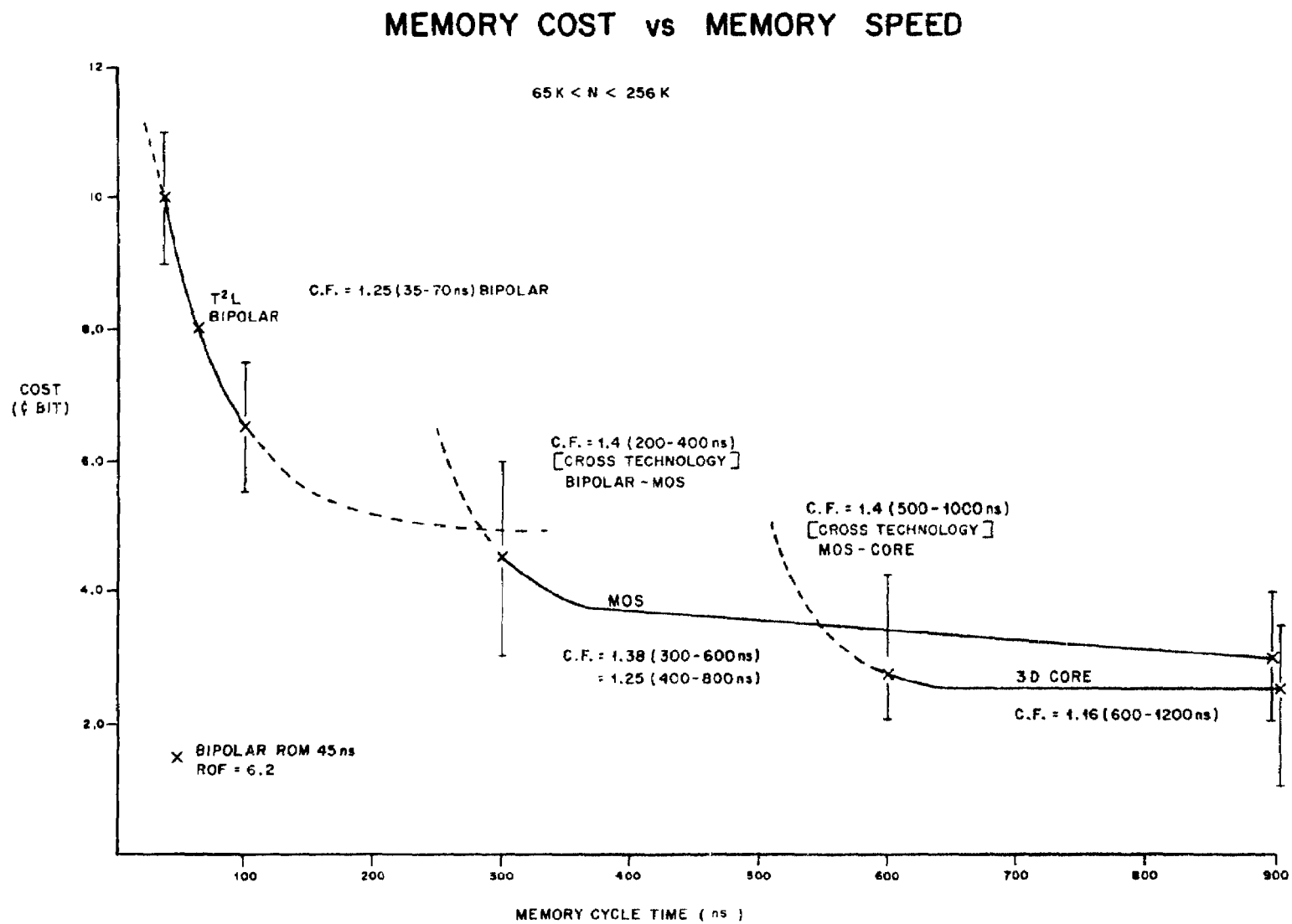


Figure 9. Memory Cost vs. Memory Speed

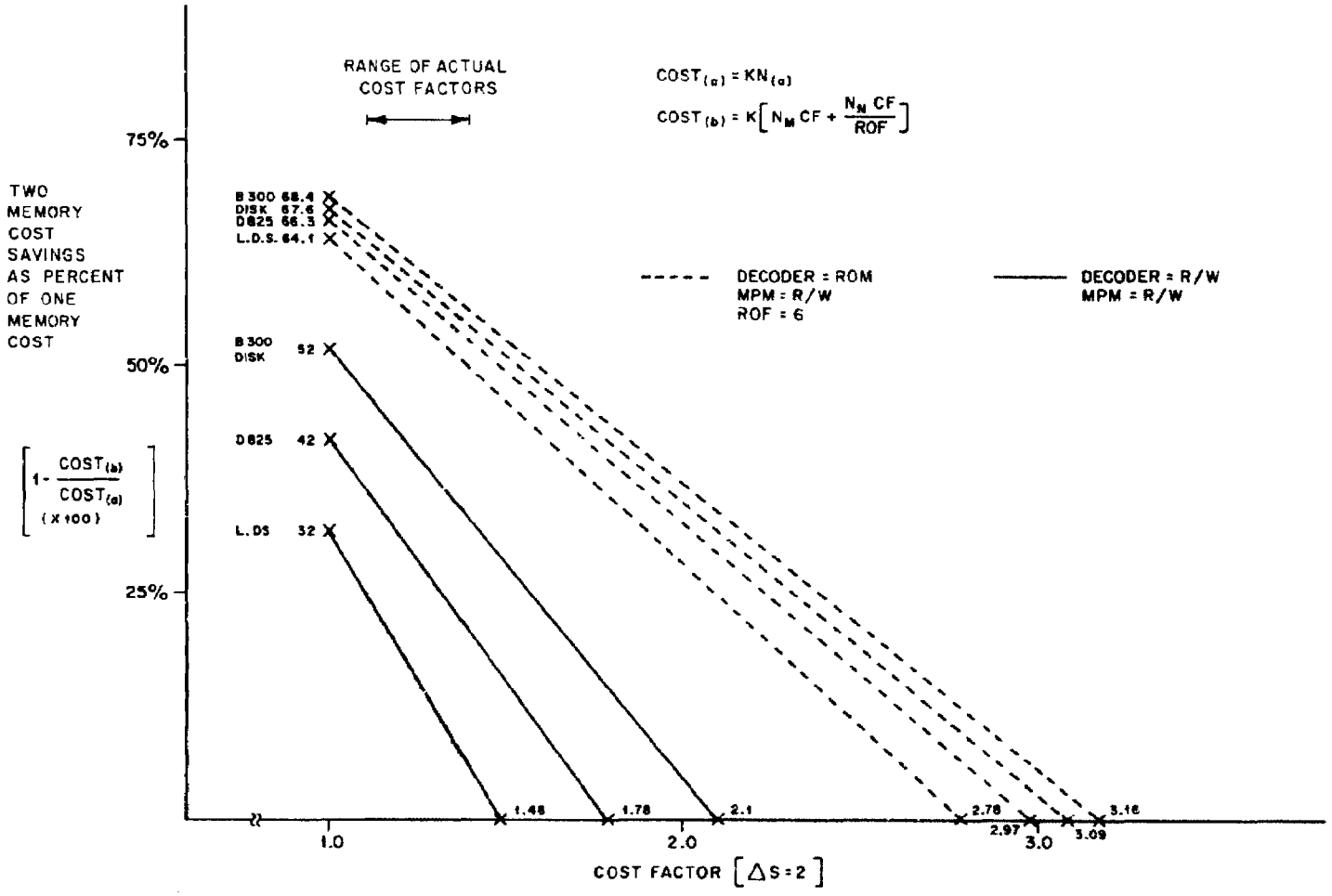


Figure 10. Two Memory Cost Savings vs. Cost Factor

in Figure 10 for the four sample programs using a "read only factor (ROF)" of 6. This ROF is an estimate of the ratio of the price of read-write memory to that for read-only memory.

In both cases, the values for a cost factor of 1.0 are the cost savings if memory cost were constant with respect to memory speed. The abscissa gives the cost factors required for the two approaches to be equal in cost.

LOADER (LDR)

One LDR is required for each Interpreter. The LDR provides clock controls for the Interpreter and the means for loading the Interpreter's MPM and Nanomemory from one of three sources:

1. Switches on the MPM/Nanomemory light panels.
2. A card reader assigned to loading.
3. The least significant 16 bits of the MIR of the same Interpreter.

It is possible to load several Interpreters concurrently from their panel switches or from their MIR's. Concurrent loading into more than one Interpreter from the card reader assigned to loading is not permitted.

Figure 11 is a diagram of the loading functions in the LSI multiprocessor.

Loading from the MIR is under microprogram control and provides the capability for programmatic overlay of the MPM and Nanomemory from any S memory module or any device attached to the Switch Interlock. A more detailed description of programmatic overlay from S memory is given in Sections VII and VIII.

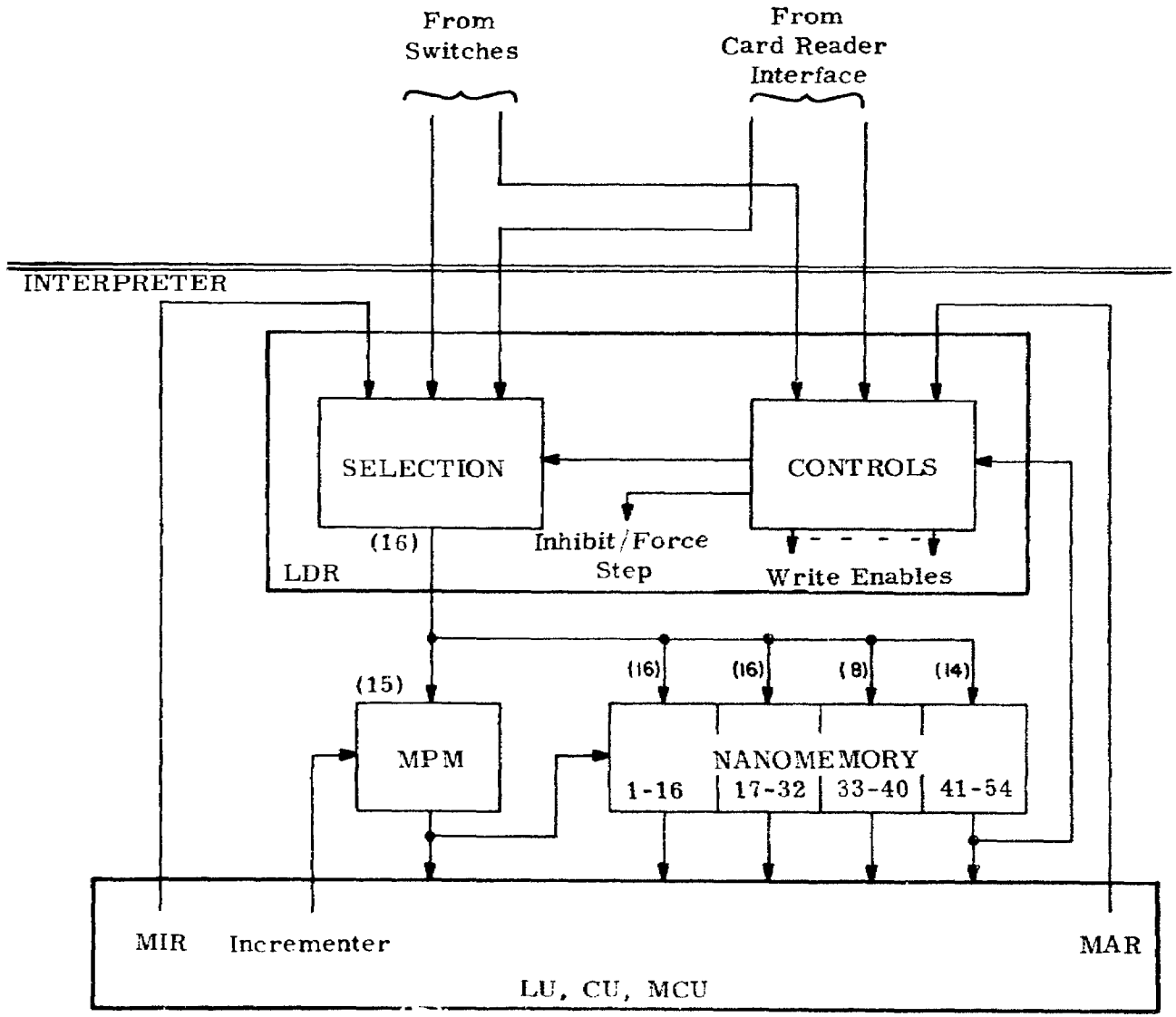


Figure 11. Implementation of Loading Functions Block Diagram

SECTION III

MULTIPROCESSING HARDWARE DESCRIPTION

MULTIPROCESSOR INTERCONNECTION

A major goal in multiprocessor system design is to increase efficiency by the sharing of available resources in some optimal manner. The primary resource, main memory, may be more effectively shared when split into several memory "modules". A technique for reducing delays in accessing data in main memory is allowing concurrent access to different memory modules. With this concurrent access capability present, an attempt is made to assign tasks and data to memory modules so as to reduce conflicts between processors attempting to access the same memory module. Nevertheless, since some conflicts are unavoidable, a second technique (reduction of conflict resolution time) is required. These two techniques are largely a function of the multiprocessor interconnection scheme which has been discussed by Curtin[2] and others.[3,4]

Figure 12 shows three basic functional interconnection schemes. These are described in more detail by Curtin.[2]

The disadvantages of the single bus approach (Figure 12) for many processors are:

1. the obvious bottleneck in information transfer between processors and memory modules due to both bus contention and memory contention

2. the catastrophic failure mode due to a single component failure in the bus.

A solution to the first problem has been to increase the frequency of operation of the bus.[2,5]

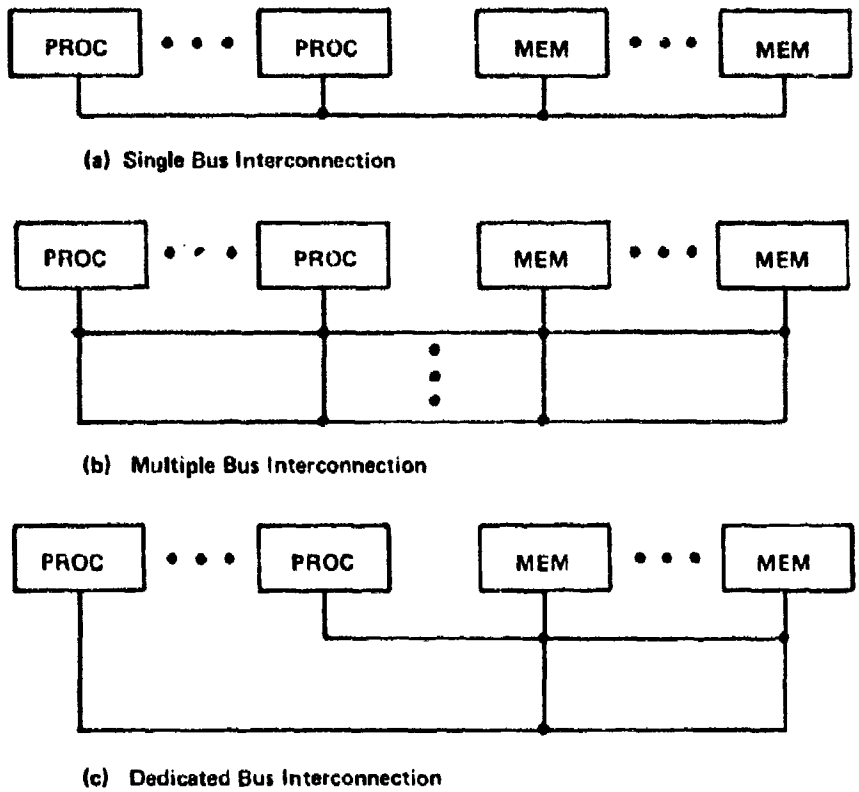


Figure 12. Functional Multiprocessor Interconnection Scheme

The multiple bus approach is merely an extension of the single bus approach where all processors contend for use of any available (non-busy) bus. The advantages are redundancy and allowing an appropriate number of buses (less than the number of processors) to handle the traffic between processors and memory modules.

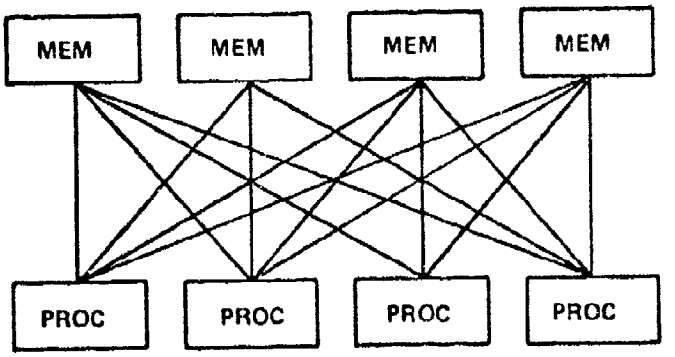
The third approach utilizes a dedicated bus structure (one per processor). Although this approach required more buses, it requires neither the logic nor, more importantly, the time for resolving priority between processors requesting the use of a bus. Proponents of this approach contend that the time penalty for resolving conflicts for access to a memory module is enough of a price to pay without having to wait for the availability of a bus.

In a Hughes report,[4] the authors distinguish the physical differences between two multiprocessor interconnection schemes. The two approaches (one called multiport and the other called matrix switch) are shown in Figure 13.

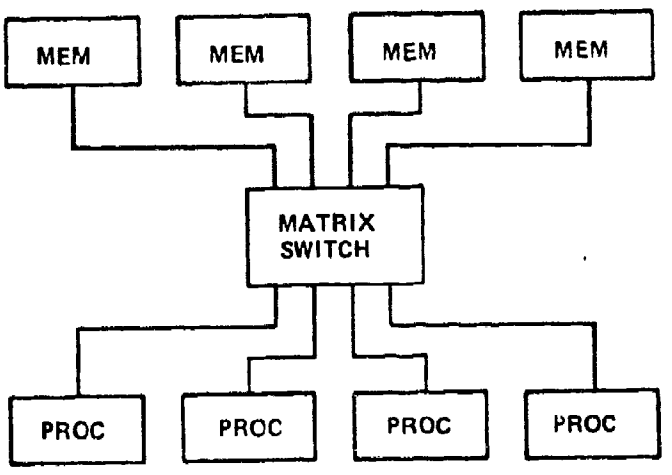
The Hughes report characterizes the two connection approaches as follows:

> "In the multiport approach, the access control logic for each module is contained within that module, and intercabling is required between each processor and memory pair. Thus, the total number of interconnecting cables is the product of the number of processors and the number of memories. Each module must be designed to accommodate the maximum computer configuration.
>
> "In the matrix switch approach, the same interconnection capability is achieved by placing the access control logic for each module in a separate module. The addition of this module to the system is compensated (for) by reducing the intercables required to the sum of the processors and memories rather than the product and by not penalizing the other modules with maximum switching logic.
>
> "There generally is no speed differential between multiport and matrix arrangements. The major difference lies in the ability to grow in wiring complexity. Multiprocessors with multiport arrangements are generally wired, at production time, to the maximum purchased configuration. Future subsystem expansion generally requires depot level rewiring. This problem generally does not exist with the matrix arrangement. The maximum capacity is wired in but the switching logic complement reflects the purchased system. Subsystem expansion entails purchase of added processor/memory modules (and necessary cabinetry if required) plus the required switch matrix logic cards."

Apparent from the arguments in this report is the desire to reduce the number of wires interconnecting the processors and memory modules. A way to reduce the wiring (in addition to the use of the matrix switch) is by using serial transmission of partial words at a frequency several times that of the processors. This technique has been used by Meng[5] and Curtin.[2] The tradeoff here is between the cost



(a) Multiport



(b) Matrix Switch

Figure 13. Physical Multiprocessor Interconnection Scheme

of the transmitting and receiving shift registers and the extra logic necessary for timing and control of the serial transmission versus the cost of wiring and logic for the extra interconnection nodes for a fully parallel transmission path.

Another factor adversely affecting efficiency in a multiprocessing system is a variation in the amount of computation versus I/O processing that must be done. In previous multiprocessing systems I/O functions and data processing functions have been performed in physically different hardware modules with devices being attached only to the I/O controllers (Figure 14). (This technique is typical of Burroughs D825, B 5500, or B 6700). In a multi-interpreter system, however, processing and I/O control functions are all performed by identical Interpreters whose writable microprogram memory can be reloaded to change their function. This technique allows a configuration (Figure 15) in which the devices are attached to the same exchange as the memories and processors.

THE SWITCH INTERLOCK

The Multi-Interpreter interconnection scheme for forming a multiprocessor is called a "Switch Interlock": a dedicated bus, matrix switch with an optional amount of serial transmission.

The Switch Interlock is a set of hardware building blocks that connects Interpreters to devices and memory modules. Connection between Interpreters and devices is by reservation with the Interpreter having exclusive use of the (locked) device until specifically released. Connection with a memory module is for the duration of a single data word exchange, but is maintained until some other module is requested or some other Interpreter requests that module.

Consistent with the building block philosophy of Interpreter-based systems, the Switch Interlock is partitioned to permit modular expansion for incremental numbers of Interpreters, memory modules or device ports and modular selection of the amount of parallelism in the transfer of address and data through the Switch Interlock from fully parallel to fully serial. Functionally, the Switch Interlock consists of: parallel-serial conversion registers for each Interpreter, input and output selection gates, parallel-serial conversion registers for each memory module and each device, and associated control logic. Figure 16 outlines the implementation of the Switch Interlock and shows the functional logic units that are repeated for each Interpreter, memory module, and device. The bit expandability of the Switch Interlock is shown by dashed lines between the input/output switches and the shift registers associated with the memory module, devices, and Interpreters.

The Switch Interlock in the LSI Multiprocessor handles five Interpreters, eight memories and eight device ports (more than one device could be attached to each port). The transmission paths through the Switch Interlock break the 32-bit data word into 8 wires carrying 4 serial bits each, transmitted with a "high speed" clock having a frequency five times that of an Interpreter clock.

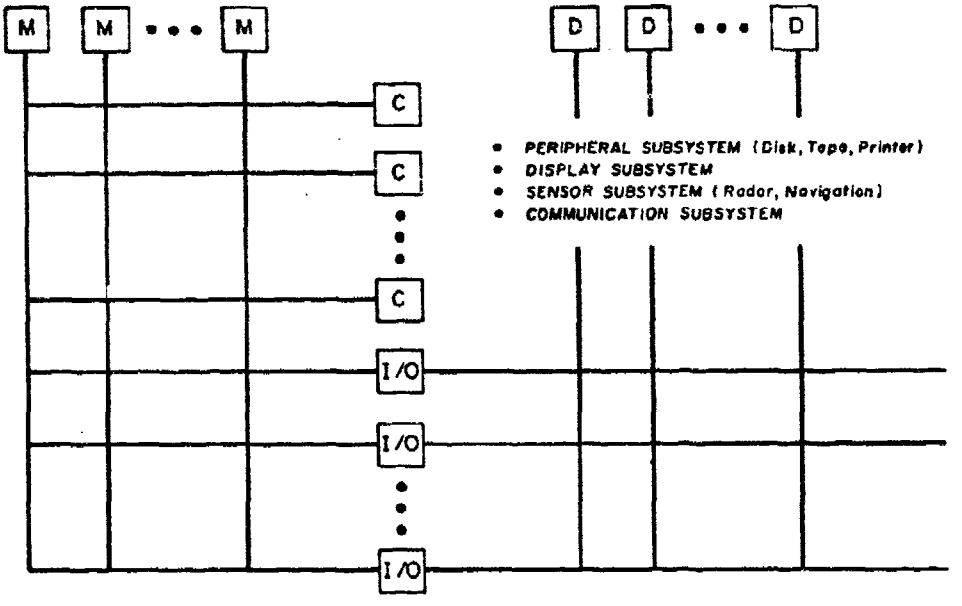


Figure 14. Centralized Multiprocessor System

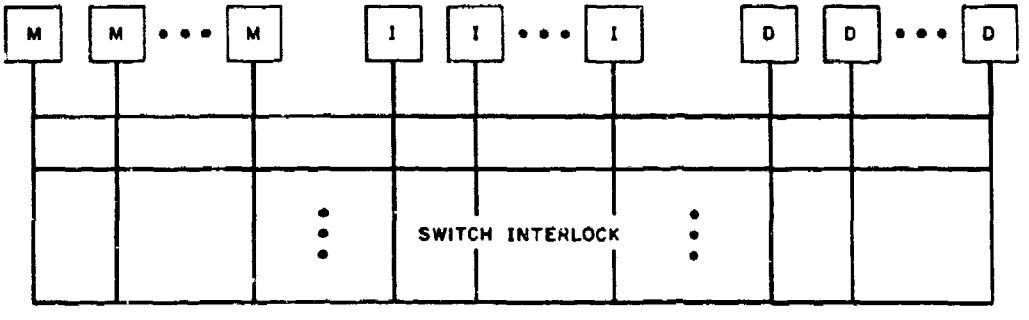


Figure 15. Distributed Multiprocessing Interpreter System

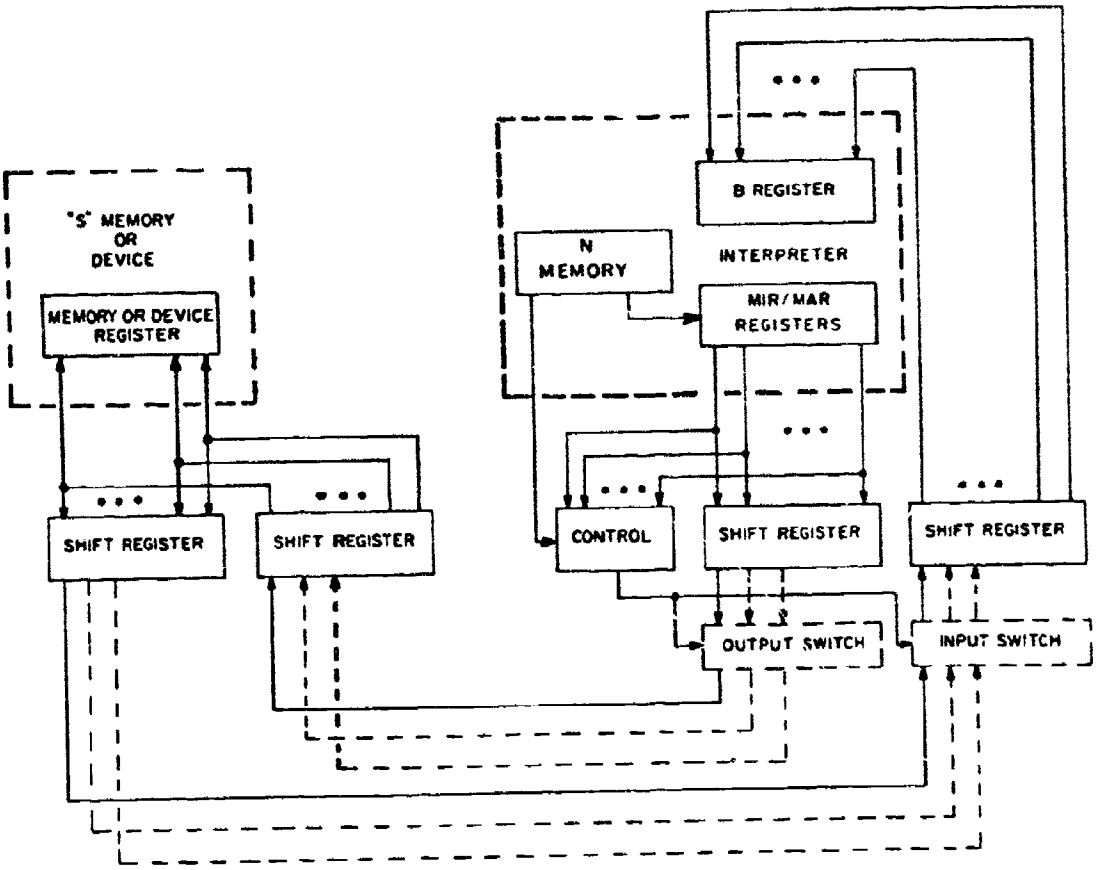


Figure 16. Implementation of the Switch Interlock

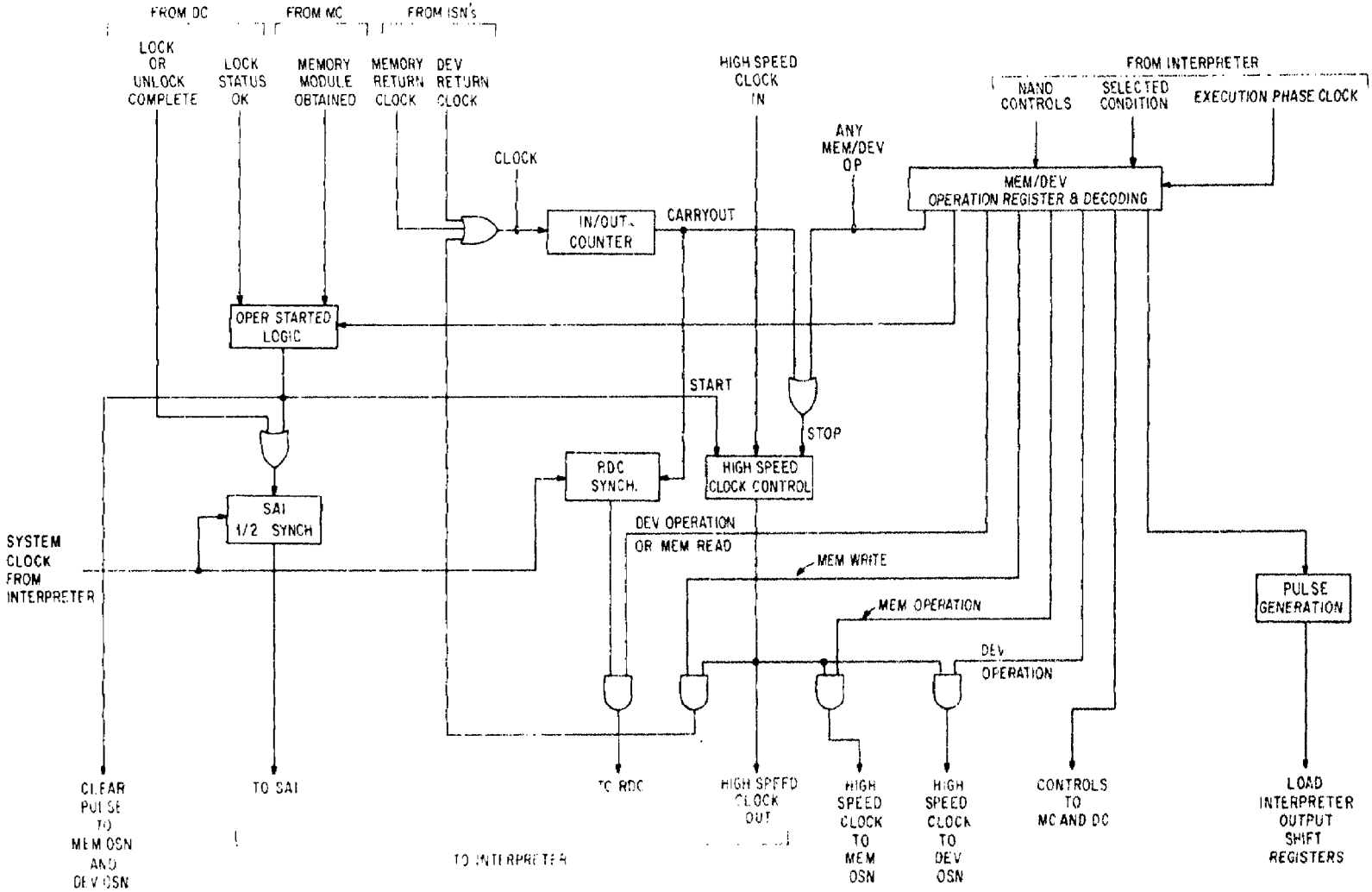


Figure 17. Memory/Device Controls (MDC) Block Diagram

The six basic modules for the Switch Interlock of the LSI Multiprocessor are described below.

Memory/Device Controls (MDC)

The MDC controls the high-speed clock used for the serial transmission of data (Figure 17) and is an interface between the Interpreter and the controls described below (MC and DC). There is one MDC per Interpreter. Physically, the MDC's for two Interpreters are contained in one finned 5-inch by 5-inch by 1/2-inch plate.

Device Controls (DC)

The DC resolves conflicts between Interpreters trying to lock to a device and checks the lock status of any Interpreter attempting a device operation (Figure 18). Physically, the DC is contained on two identical finned plates, each plate capable of handling up to three Interpreters and up to eight devices. System expansion using this module could be in number of Interpreters or in number of devices.

Memory Controls (MC)

The MC resolves conflicts between Interpreters requesting the use of the same memory module (Figures 19 and 20). Physically, the MC is contained on two finned plates. One plate contains the MC for three Interpreters and eight memory modules and the other plate contains the MC for the other two Interpreters and eight memory modules, plus the "memory-busy" flip-flops. The global condition bit priority resolution and the interrupt Interpreter logic is also physically located on this second plate although it is functionally independent. System expansion using the MC could be in number of Interpreters or in number of memory modules.

Output Switch Network (OSN)

The OSN sends data, address, clock, and control from Interpreters to addressed devices or memory modules (i.e., the OSN is a "demultiplexer"). Physically, the OSN is made of two different types of finned plates handling either three or four wires for up to five Interpreters and eight devices or memory modules. One type of plate handles four data-type paths for five Interpreters and eight devices or memories. The other type of plate handles two data-type paths and one clock-type path for five Interpreters and eight devices or memories. Logic diagrams of these types of OSN's are shown in Figures 21 and 22. Each column of logic is for one Interpreter with the inputs from the Interpreter coming in the top. Each row represents one serial transmission path and the outputs to eight devices or memories coming from the side and bottom of the drawing. System expansion using these modules could be in number of

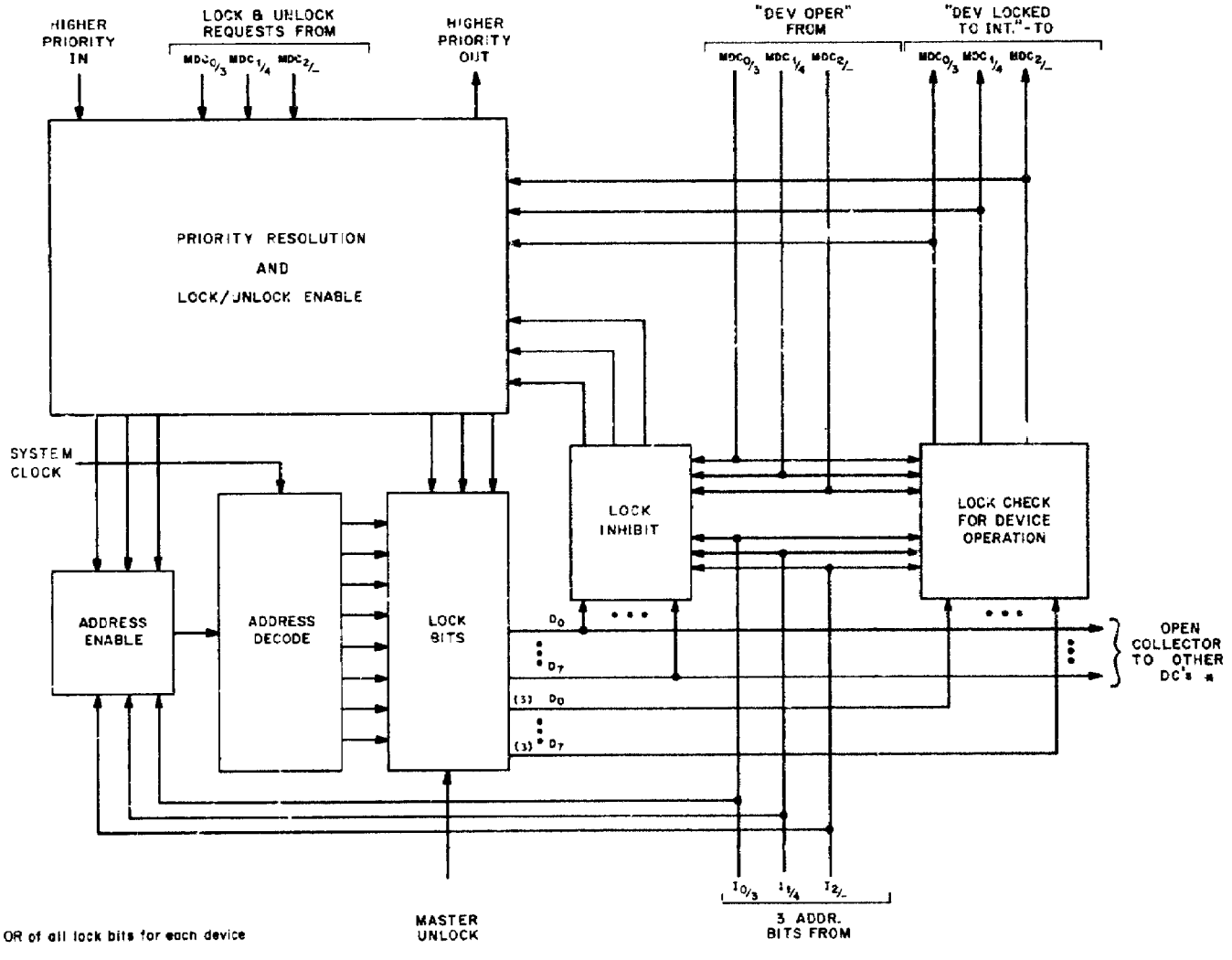


Figure 18. Device Controls (DC) Block Diagram

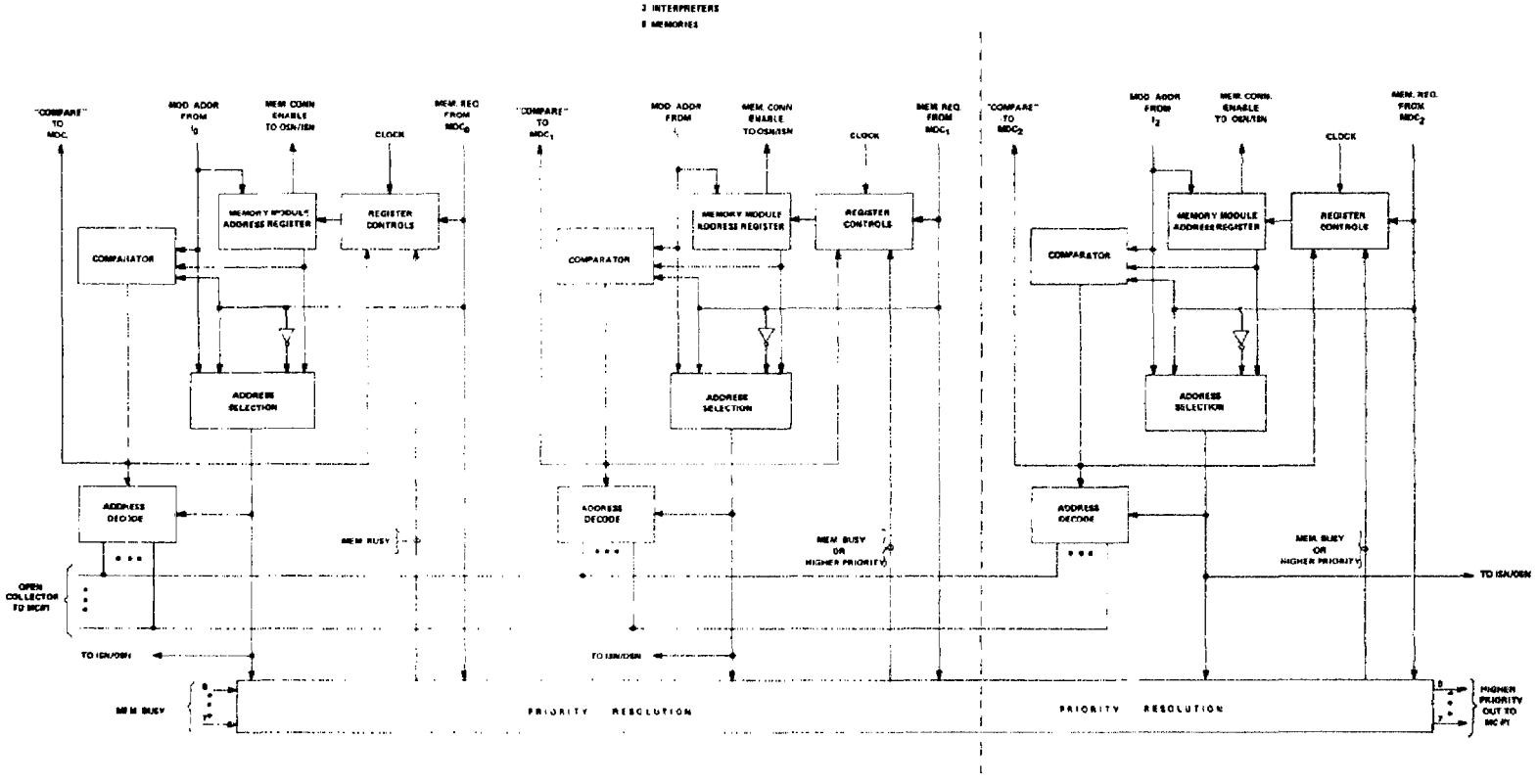


Figure 19. Memory Control No. 0, Block Diagram

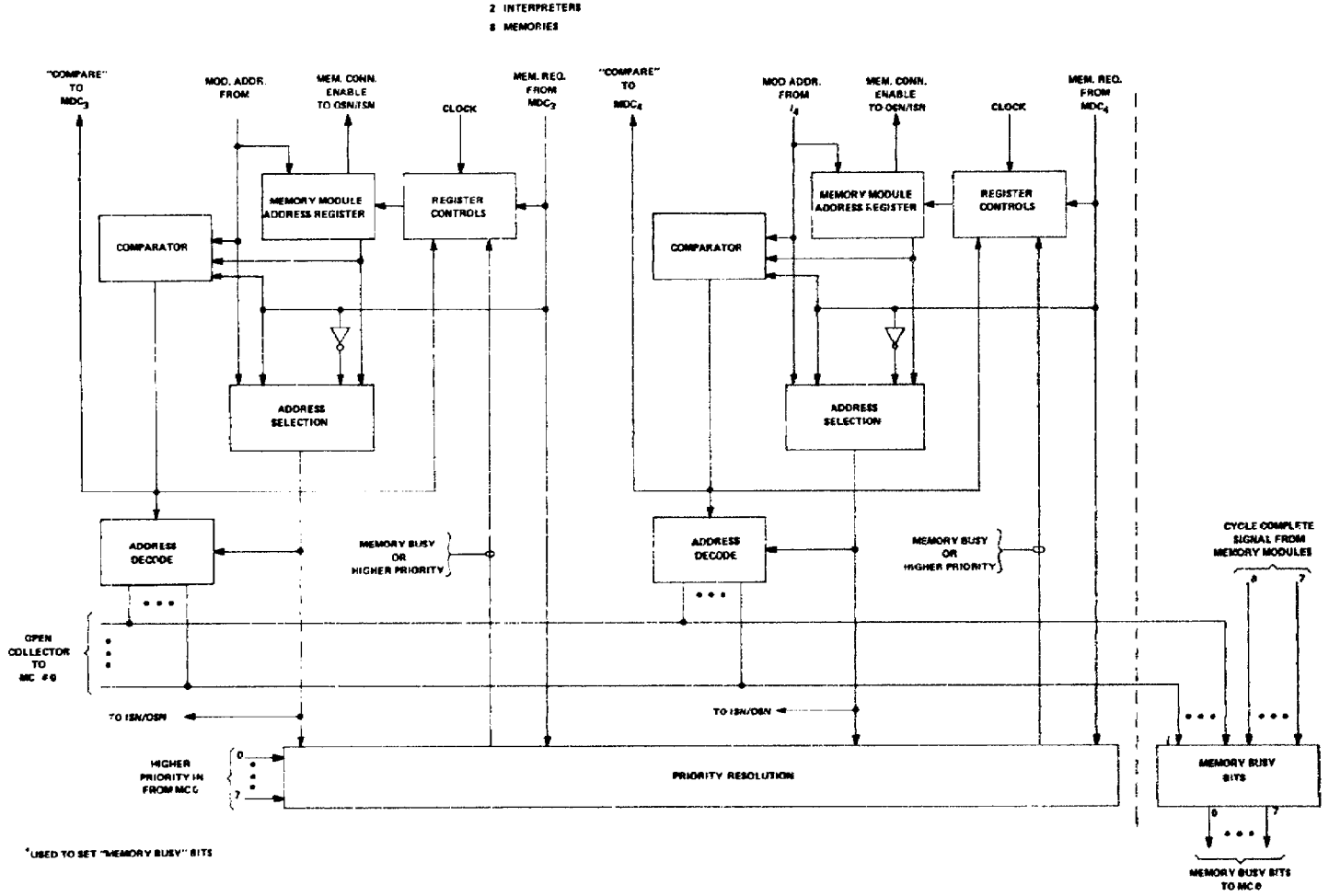


Figure 20. Memory Control No. 1, Block Diagram

Interpreters or in number of devices or memories. The number of replications of this plate would also change if the amount of serialization of the data path were changed.

Input Switch Network (ISN)

The ISN returns data from addressed devices or memory modules to the Interpreters (i.e., the ISN is a "multiplexer"). One finned plate handles five wires for five Interpreters and up to eight devices or memory modules. A logic diagram for the ISN is shown in Figure 23. As with the OSN, each column of logic is for one Interpreter with the outputs to the Interpreter coming from the top. Each row also represents one serial transmission path with the inputs from eight devices or memories coming in the side of the drawing. System expansion using this module could be in number of Interpreters or in number of devices or memories. The number of replications of this plate would also change if the amount of serialization of the data path were changed.

Shift Register (SR)

These units are parallel-to-serial shift registers or serial-to-parallel shift registers that use a high frequency clock for serial transmission of groups of four data and address bits through the ISN's and OSN's. They are physically located with the Interpreters, device interfaces, and memory module interfaces.

POWER DISTRIBUTION

Figure 24 shows the details of the power distribution system in the aerospace multiprocessor. Even though all a-c connections are shown schematically attached to one line, a load center is mounted inside the cabinet and two phases of a three phase four wire 120/208 volt 60 Hz input are each connected through the load center to four strips of electrical outlets mounted inside the cabinet.

As shown, each Interpreter has its own power supply with a connection to the Switch Interlock for supplying +5 volts to the MDC for that Interpreter. All +5 volt distribution is by heavy gauge wire twisted with its return. All sensing and connections of return to chassis are done at the point of load. The system power supply provides power to the device and memory interfaces, the real time clock, power control and clock distribution, the light panel, and the Switch Interlock. The sensing for the system power supply is on the Switch Interlock.

As can be seen, the multiplicity of reference-to-reference connections via the cold side of the twisted pairs made proper "treeing[6]" of the references before connection to earth impractical. Therefore freely tying reference to chassis was allowed.

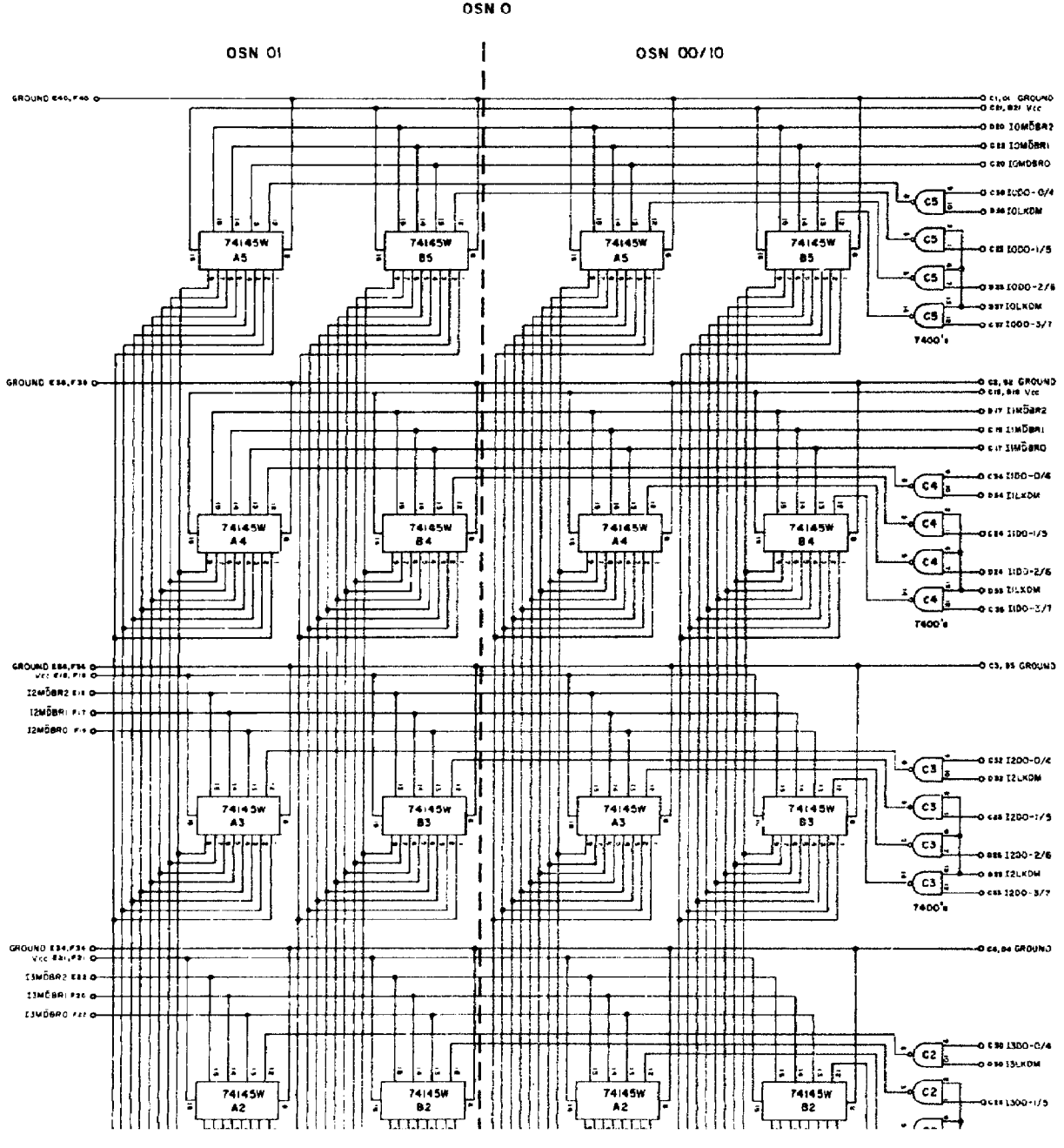
In retrospect, the only changes suggested would be providing a better reference-to-reference connection between each Interpreter and the Switch Interlock, and removing the reference to chassis connections on the +12 volt, -12 volt, and +20 volt supplies after insuring a suitable reference to chassis connection at the loads.

The only grounding problem encountered was on the loader board in the Interpreters. This problem was eliminated by installing a wire ground grid on the board and by providing extra ground pins from the board to the backplane. Of interest is that no decoupling capacitors exist in the system. Space for decoupling capacitors has been provided and should be added if noise problems are encountered; however no such problems have arisen during the fairly extensive testing before and after delivery.

CLOCK AND POWER CONTROL

From the description of the Switch Interlock, it is clear that two clocks having different frequencies are needed in the aerospace multiprocessor. During the design of the aerospace multiprocessor the relationship between the maximum shift rate through the Switch Interlock and the maximum speed of the Interpreters was determined to be at least 4:1. Since four bits are transmitted serially on each path through the Switch Interlock and shifting is to be finished within one Interpreter clock time, a ratio of 5:1 was selected. However, from the implementation as shown in Figure 25, this ratio could be easily changed by changing the value preset into the counter. The logic appearing in this figure is all controlled by a central system power supply, which in a failsafe system must be made redundant.

As shown in the figure, the width of the high-speed clock to the MDC's in the Switch Interlock is controlled by the width of the master clock coming in from the pulse generator, and the width of the Interpreters' clock is controllable by varying the resistor value on the single shot. The flip-flop control has been added to the clock for each Interpreter to insure against performing any spurious memory or device operations while power is either being applied or being shut off


OSN 0
OSN 01
OSN 00/10
74145W A5
74145W B5
74145W A4
74145W B4
74145W A3
74145W B3
74145W A2
74145W B2
7400's

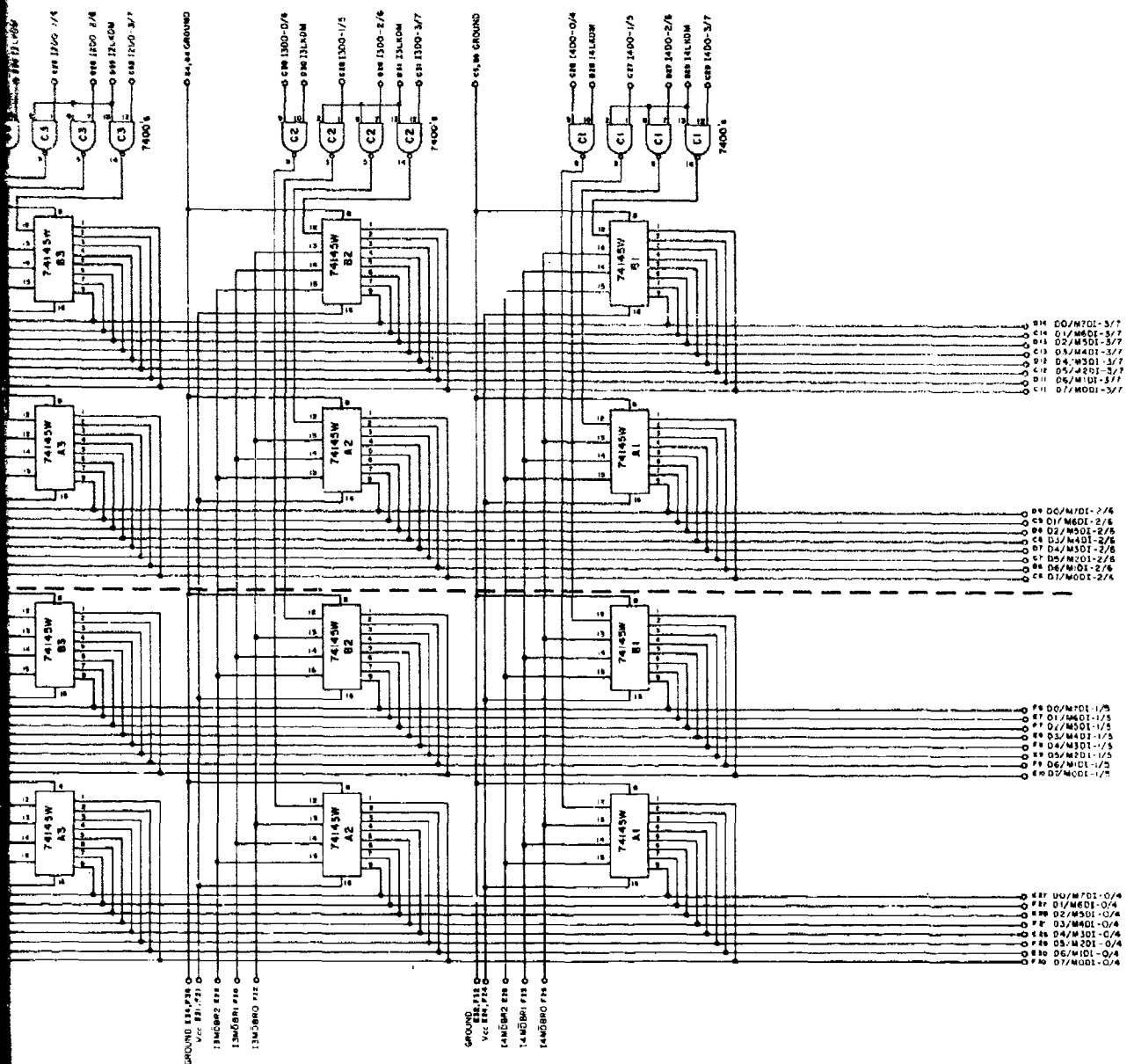
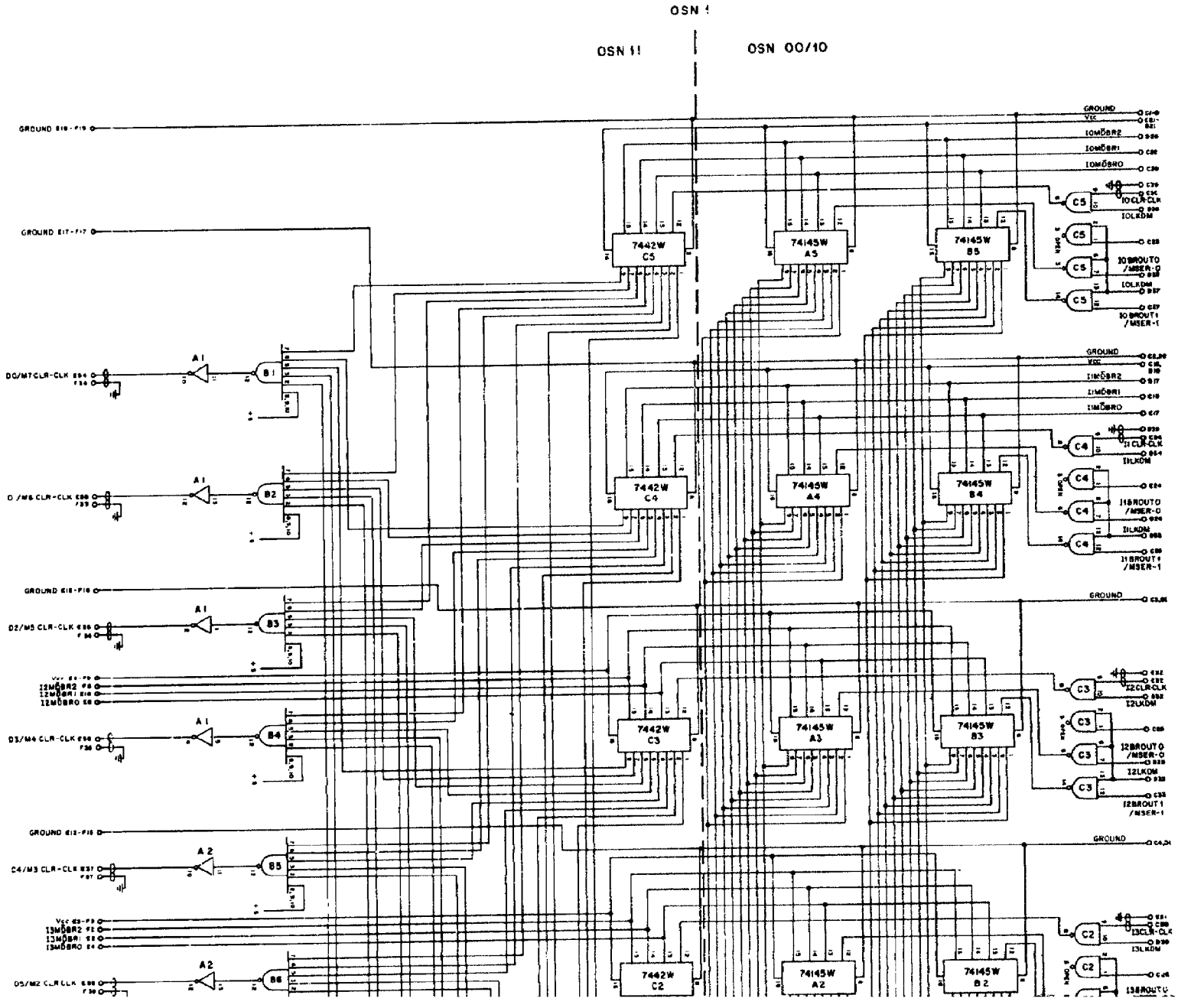


Figure 21. Output Switch Network No. 0, Logic Diagram



NOTE A1 - A2 ARE 7418 OSN II
B1 - B6 ARE 74S03 OSN II
C1 - C5 ARE 74S08 OSN 00/10

Fig

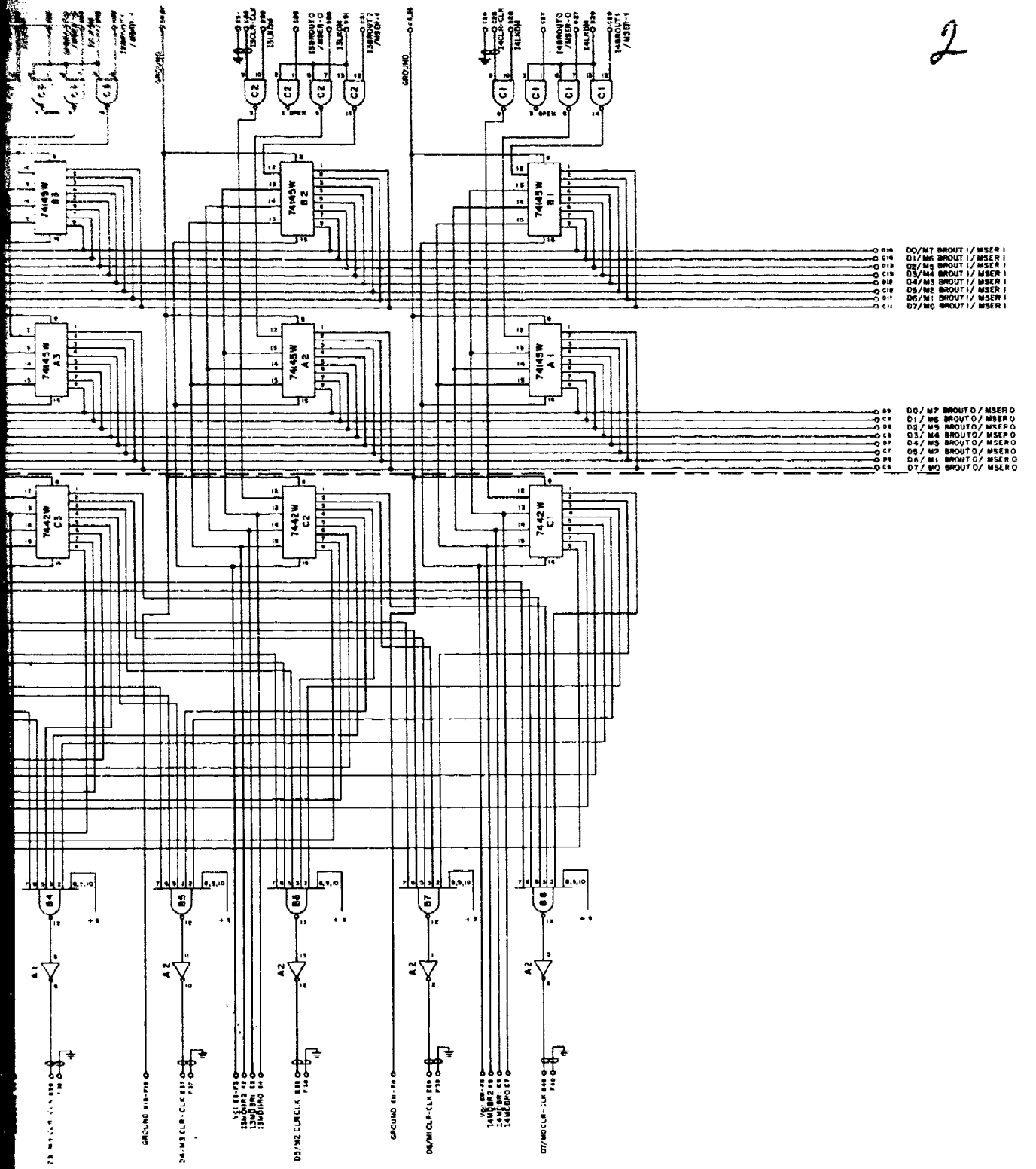
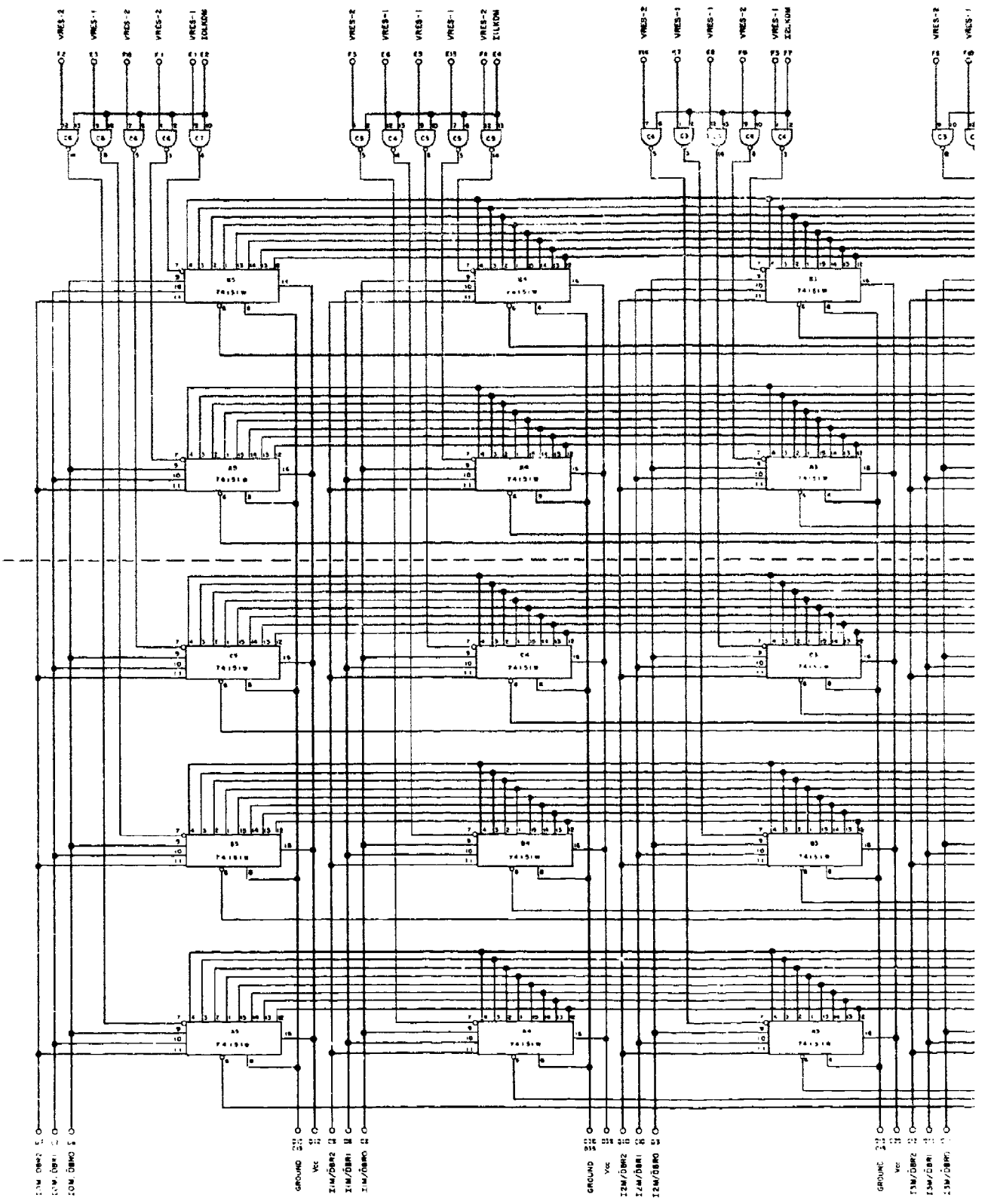


Figure 22. Output Switch Network No. 1, Logic Diagram

1



NOTE ISM01 HAVE COMMON GROUNDS AND VOLTAGES THAT ARE DIFFERENT THAN THE OTHER PACKAGES.

Vcc - E16
GND E23, F23

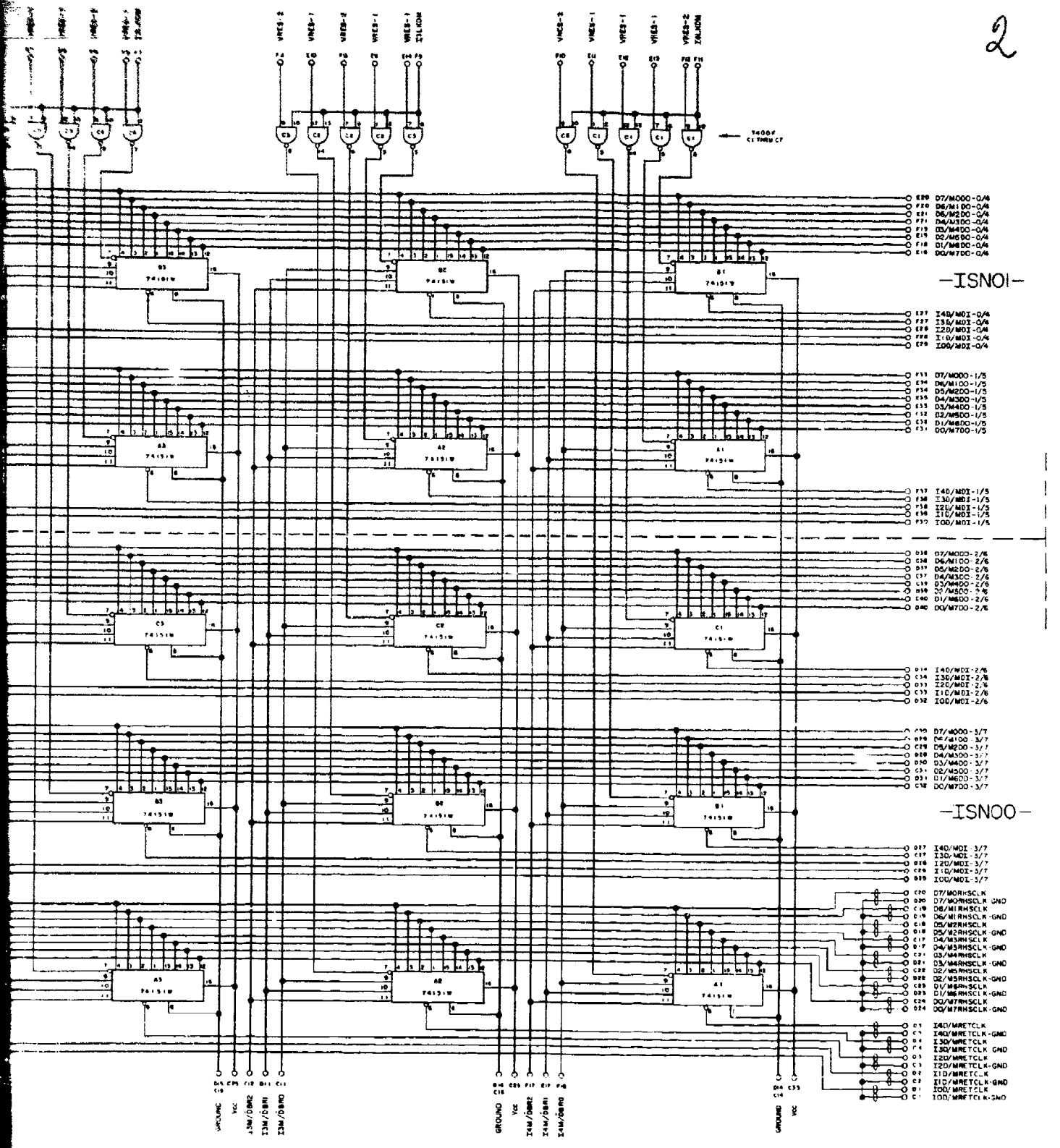


Figure 23. Input Switch Network No. 1, Logic Diagram

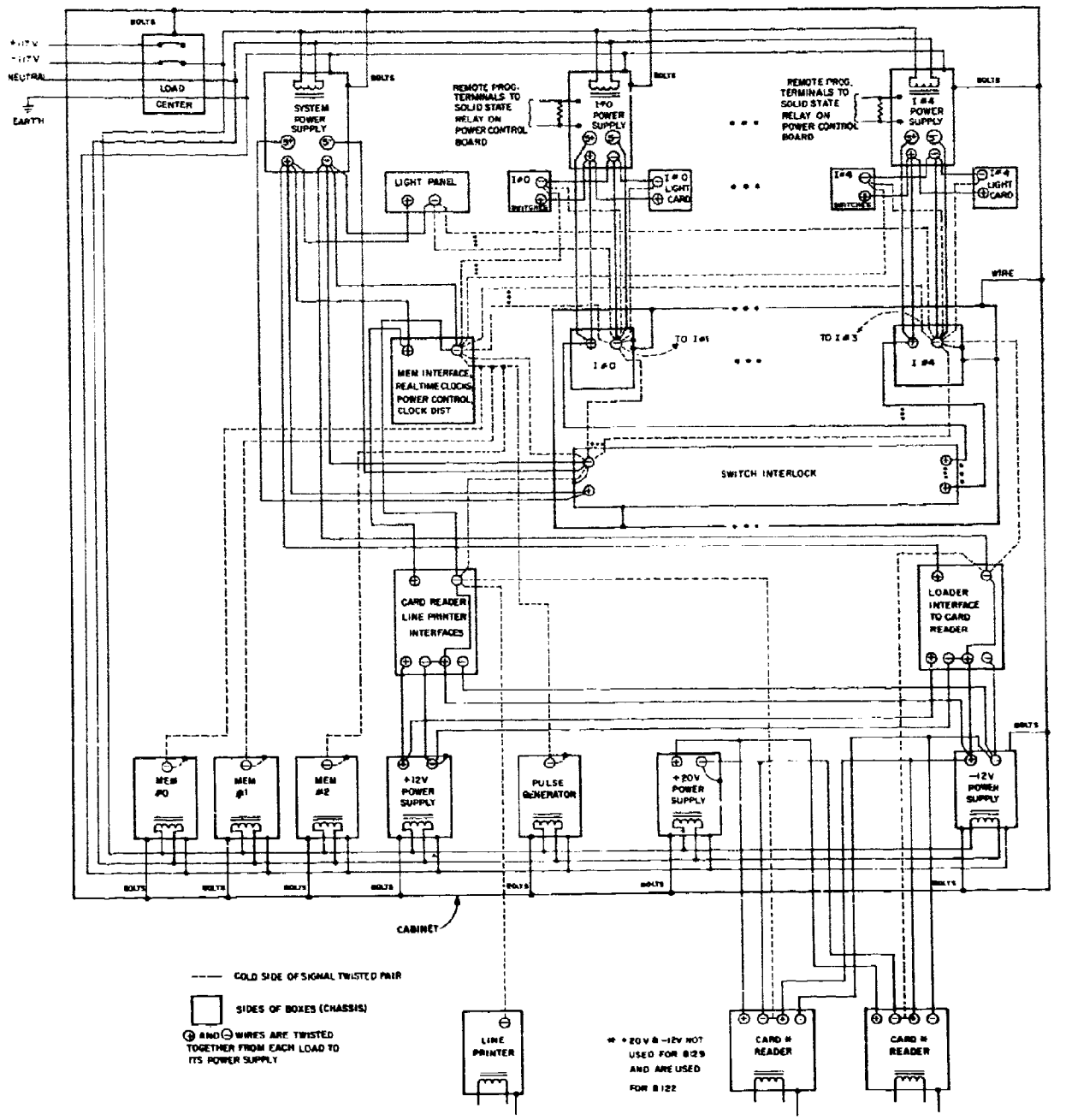


Figure 24. Power Distribution System

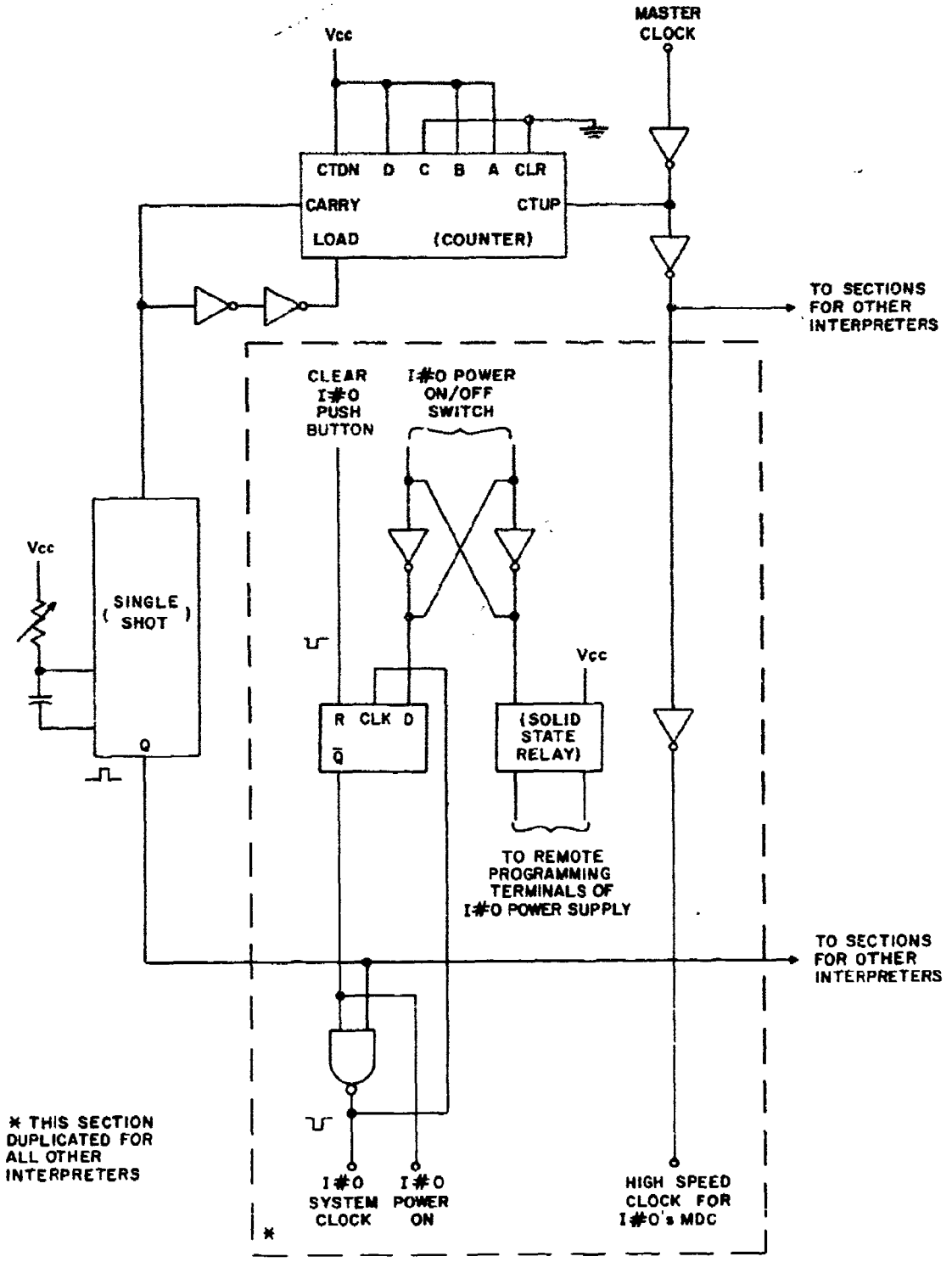


Figure 25. Implementation of Multiprocessor Clocks

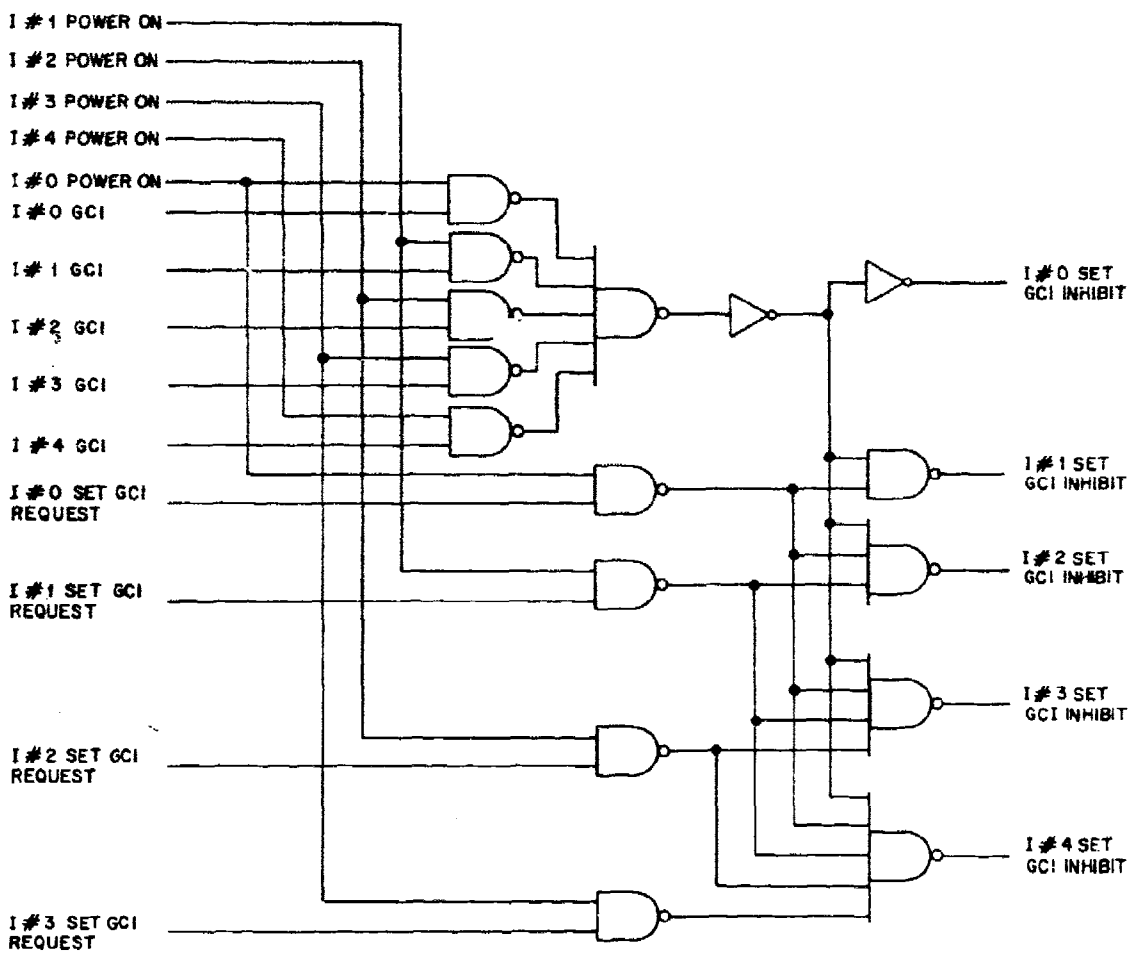


Figure 26. Conflict Resolution Logic for Global Condition Bit GC1

to an Interpreter. This is done by a front panel switch setting the flip-flop (which will shut clocks off) and turning the solid state relay on, which will then short across the resistor on the remote programming terminals of the Interpreter's power supply, turning the power supply off. When the front panel switch is set to turn power back on, the solid state relay will turn off, opening up the output of the relay and turning power back on to the Interpreter. However, if clocks were applied at this time, they would start during the powering up of the Interpreter and would continue even though no valid information existed in the Interpreter's Microprogram and Nano memories.

To avoid this problem, clocks are not restarted until the Clear pushbutton is pressed on the front panel, which is done in conjunction with pressing the Load pushbutton for loading the Microprogram and Nano memories from the loader card reader. Since during loading, a pseudo Type II instruction is forced by the loader, no clocks will be present to initiate any memory or device operations until loading is completed and the microprogram just loaded begins execution.

GLOBAL AND INTERRUPT CONDITION BITS

The two global condition bits in each Interpreter are used by programmatic convention for locking out other Interpreters during a read-modify-write to system tables resident in S memory. This is done independently for each of the two condition bits by not allowing an Interpreter to set its condition bit if any Interpreter's condition bit is already set or if a higher wired priority Interpreter is requesting to set its condition bit at the same time. This was initially to be done by chaining the priority through the Interpreters so that no external logic would be required. However, if an Interpreter's power were turned off, the chain would be broken and the same global condition bit in two Interpreters could have been set. To avoid this problem the global condition bit and the requests to set the global condition bits are brought from each Interpreter to a centralized location. (The Switch Interlock was chosen, although this logic is totally independent of the Switch Interlock operation.) In this centralized location, the power-on signals shown previously in this section are used to allow only signals from powered-on Interpreters to participate in the conflict resolution. This conflict resolution logic is powered by the system power supply and in turn sends enables back to the Interpreters for setting the global condition bits. This conflict resolution logic is shown for one of the global condition bits (GC1) in Figure 26. The same logic is repeated for the other global condition bit (GC2).

The Interrupt Interpreters condition bit, although having no priority logic associated with it, has the similar problem of having a signal from an Interpreter that is either powered down or whose power is undergoing a transition, setting the Interrupt condition bit in other Interpreters in an uncontrolled manner. To avoid this, the Interrupt signal and its control coming from each Interpreter are gated against the power-on signal for that Interpreter. These signals are then all ORed together and sent back to all Interpreters. This logic (shown in Figure 27) is also located in the Switch Interlock and is powered by the system power supply.

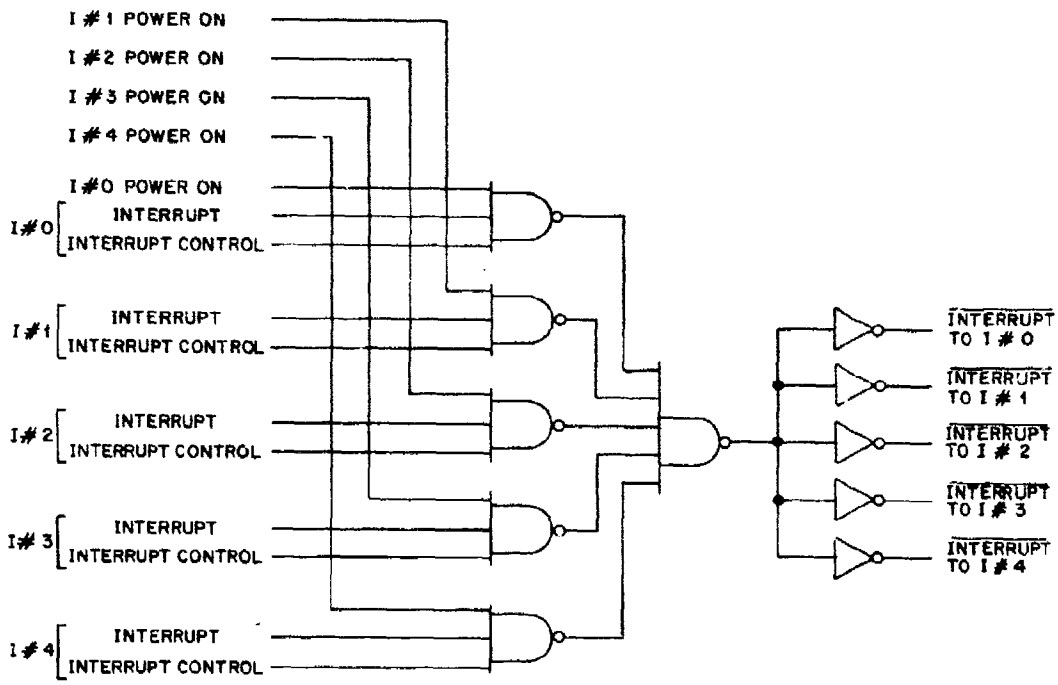


Figure 27. Implementation of Interrupt Controls

REAL TIME CLOCK AND THE HORNS

One device (device number zero) has been permanently assigned to a device called a "real time" clock, which is used programmatically to determine the failure of a task running on an Interpreter. This use is explained more fully in the Multiprocessing Control Program and Demonstration Programs section of this report. This device is merely a 32-bit counter that is counted up at a rate of once each 256 Interpreter clocks. It is powered by the system power supply and runs continuously. This device is read just as any other device attached to the Switch Interlock and must be locked to in order to be read. Since programmatically this counter is used as an interval timer, a potential problem exists if the interval to be timed were started just prior to this device overflowing (once every 2^{40} Interpreter clocks). This can be avoided by forcing the programs to test the value of the counter to insure it will not be reset during the interval of interest.

Also physically located with the real time clock are five, 4-bit counters, one associated with each Interpreter. These counters, called horns, if not reset, will overflow after every 2^{20} Interpreter clocks (approximately every 1 second for a 1 MHz Interpreter clock rate). These counters detect an Interpreter waiting for a response from a memory or device that has failed. An overflow from one of these counters will force a one clock time STEP and will set a condition bit in its associated Interpreter which then can be tested by the Interpreter. To avoid continual setting of this bit, each counter is reset every time its associated Interpreter does any memory or device operation. These operations should occur often in any program except perhaps during internal Interpreter diagnostics. These diagnostics should not require 2^{20} Interpreter clocks to run but if they did the horn for the Interpreter may be manually turned off.

INTERPRETER NUMBER

Each Interpreter is logically identical to all other Interpreters. A multiprocessing control program, however, must have a means of distinguishing between Interpreters. This is accomplished by wiring the most significant four bits of the next to the most significant 8-bit byte of the Z-input to the adder, to the connector to which the loader cable is attached. Ground and +5 volts are also wired to this connector. Within the other side of the connector, which is part of the loader cable, ground and +5 volts are jumpered to the 4 bits of Z input to appropriately indicate the Interpreter number, right justified within the 4-bit field.

SECTION IV

AEROSPACE MULTIPROCESSOR PACKAGING DESCRIPTION

MECHANICAL DESIGN

The aerospace multiprocessor is housed in a cabinet consisting of two bays 21 inches wide by 25 1/2 inches deep by 68 inches high (Figure 28). The Interpreters, and Switch Interlock modules are built up of mechanically similar submodular sections. The S memory module and power supplies are commerically available rack mounted units.

Each of the modules is made up of several finned aluminum castings (Figure 29) with massive heat sinks for mounting of the printed wiring boards and direct heat sinking of the LSI packages. Modification of the finned aluminum casting allows direct heat sinking of conventional dual in-line packages for the MPM and Nanomemories. The 5-inch by 5-inch by 1/2-inch thick submodule houses two LSI chips, as many as 98, 16-lead flat packs or as many as 45, 16-pin dual-in-line packages, depending on its function in the system.

Each of the Interpreter modules (Figure 30) and the Switch Interlock module is packaged complete with its own backplane and I/O connectors to simulate remote physical distribution of the modules.

To maintain a close physical arrangement with simulated module distribution, all of the Interpreters are mounted on a common mechanical structure which allows the multiprocessor to be mounted as a single unit on a shelf extending at right angles to the front of the two electronics cabinets, as shown in Figure 1. The multiprocessor is mounted on a swivel to allow direct access to the wire wrapped backplane during debugging and testing procedures.

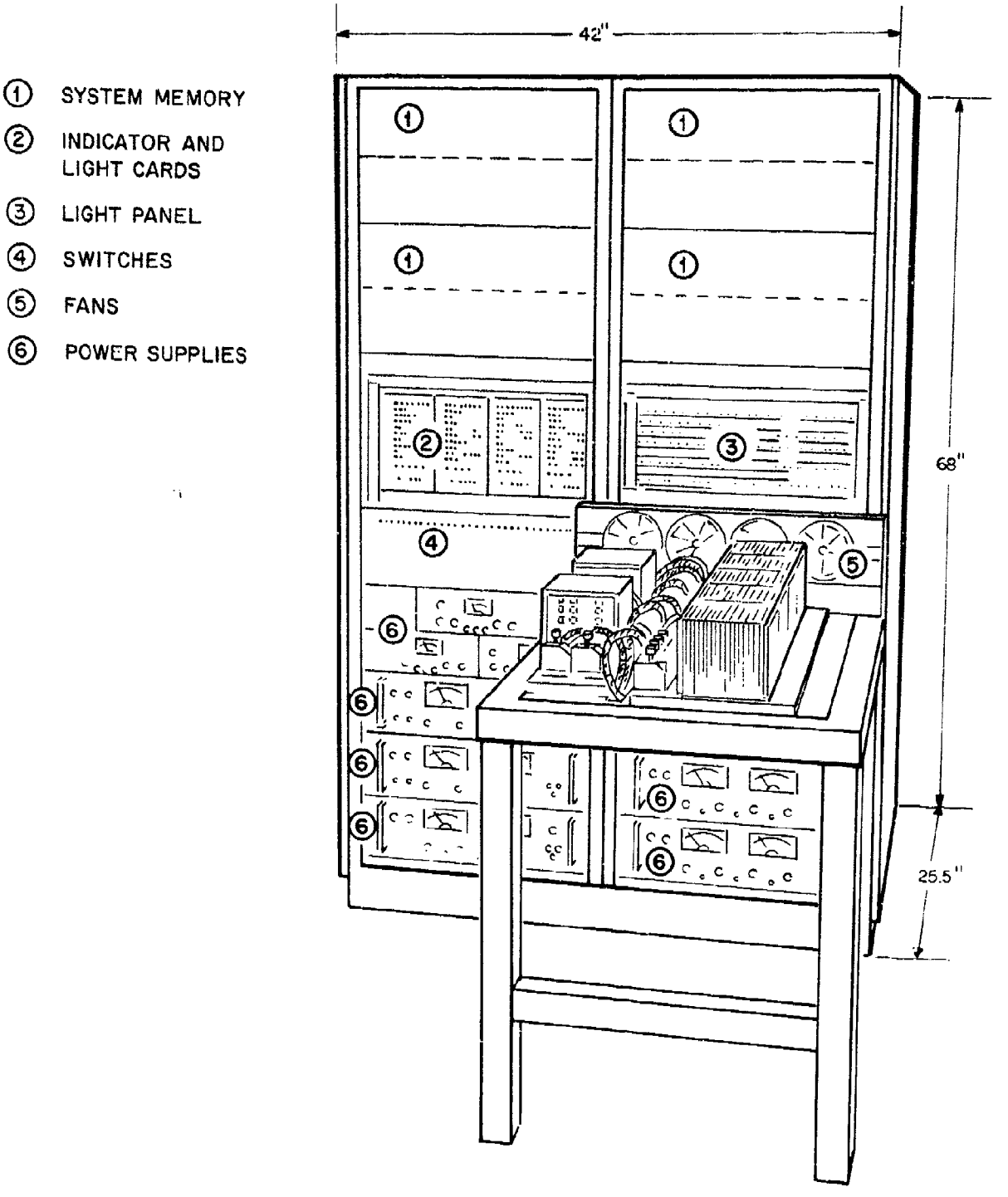


Figure 28. Aerospace Multiprocessor Configuration

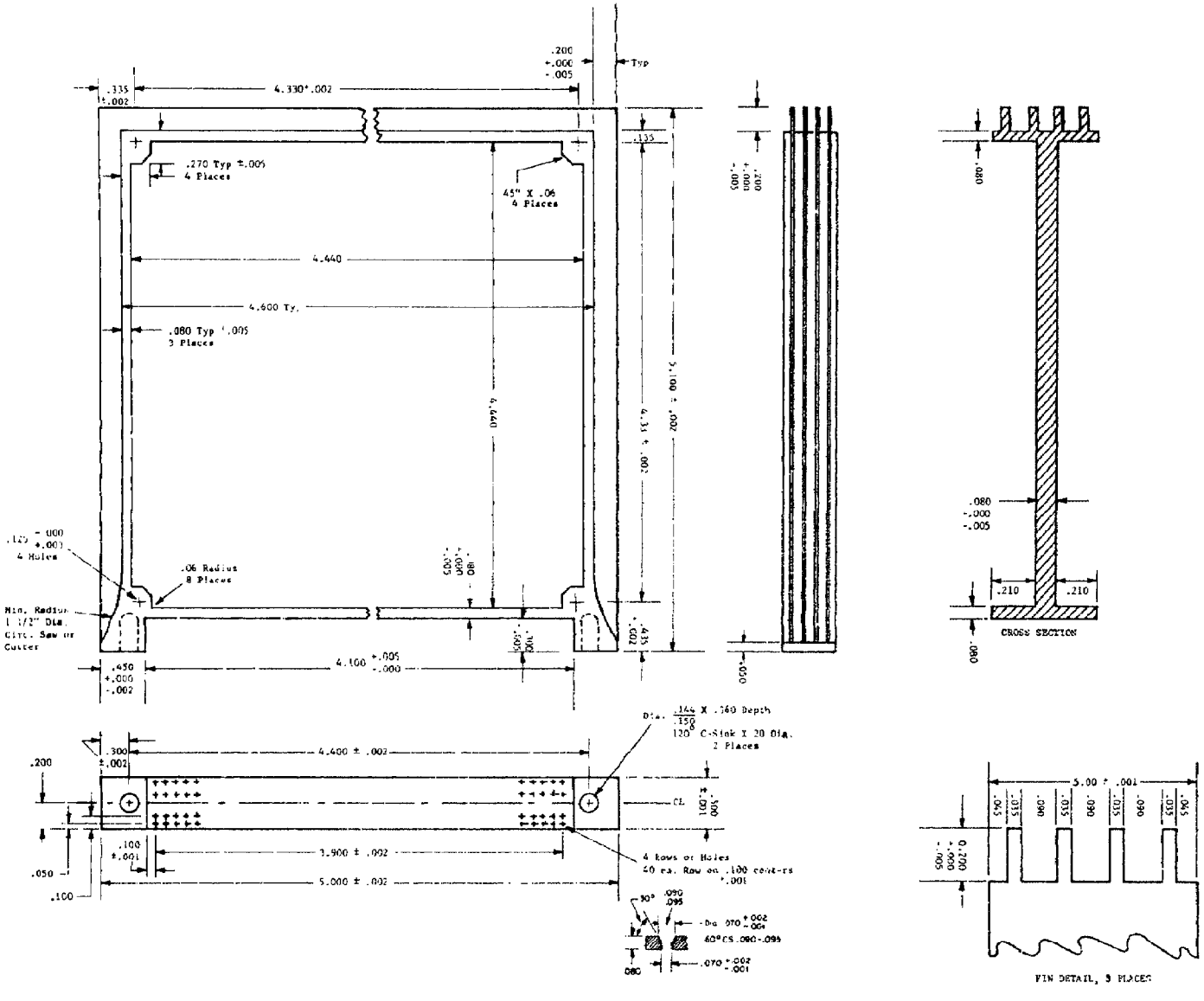


Figure 29. Submodule Housing

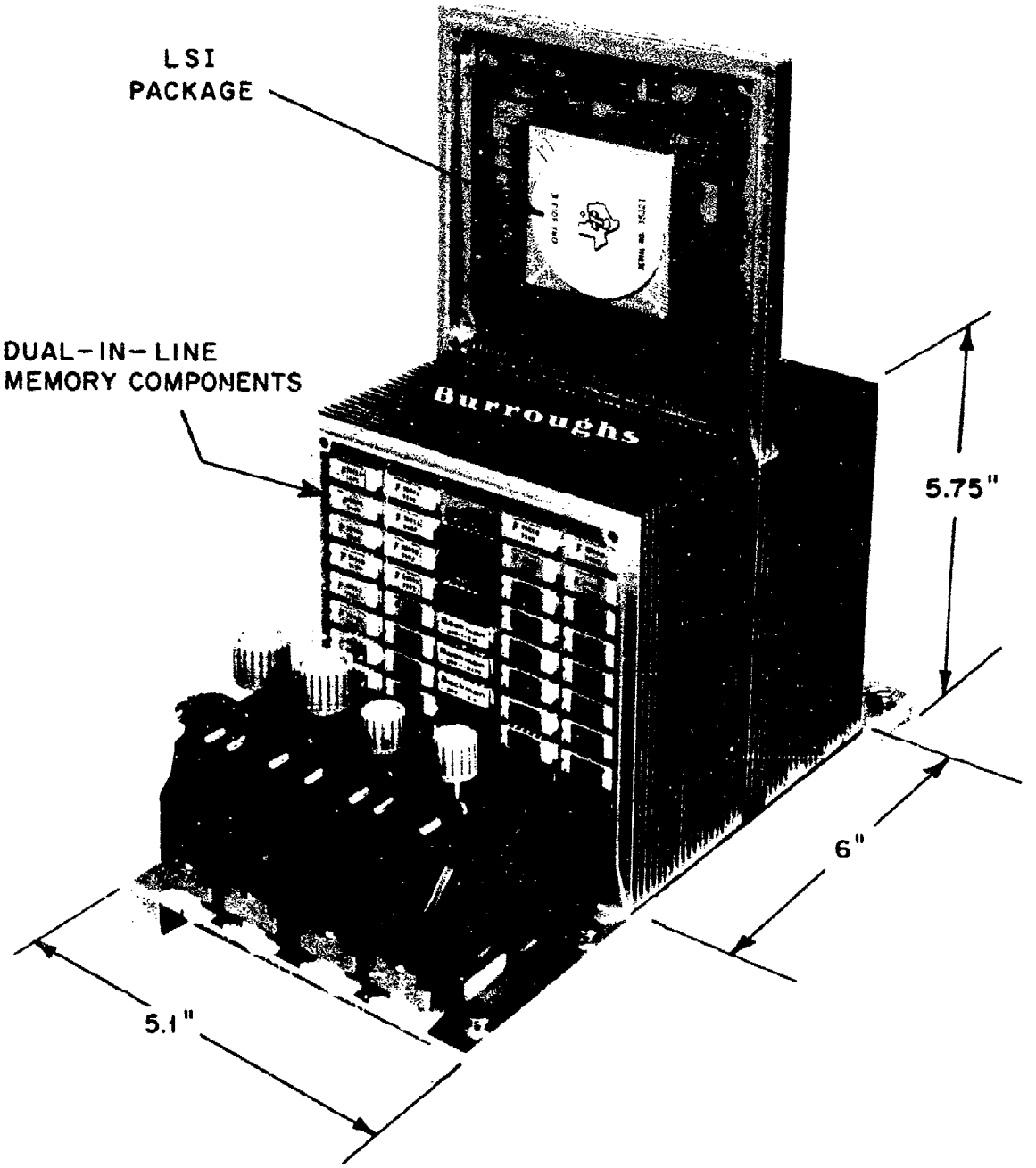


Figure 30. Interpreter Module Packaging

Figure 31 is a photograph of the system as installed at Wright-Patterson Air Force Base.

Figure 32 is a plan view of the Interpreters, Switch Interlock and connectors for interconnection among the modules.

CIRCUIT CONFIGURATIONS

The LSI multiprocessor system is implemented with the three types of submodules. The Microprogram and Nano memories in the Interpreter both use Fairchild 93410 ceramic dual-in-line packages, each containing 256 words × 1 bit of memory, interconnected with a four-layer printed circuit board mounted on the opposite side from the packages as shown in Figure 33. Since the selection of this package, Fairchild has introduced the 93415, a 1024 word × 1 bit memory package with approximately the same power dissipation as the 93410. This more dense memory package is recommended for future Interpreter systems.

The Loader submodule in the Interpreters and all submodules in the Switch Interlock use standard 54/7400 series flat packs which are mounted on either two or four layer printed circuit boards which are then mounted on the two sides of the aluminum plate submodule as shown in Figure 34. The packing density of the flat packs is typically between 25-30 per board, since most of these submodules are pin limited and would have required six to eight layer boards to achieve the maximum packing density of 49 flat packs per board.

The remainder of the Interpreter logic is implemented with Texas Instruments discretionary wired, transistor-transistor logic (TTL) using their "N" and "S" arrays as follows:

8-bit Logic Unit (two Type "N" slices)

Memory Control Unit (two Type "N" slices)

Control Unit (two Type "S" slices)

This type of submodule is shown extended above the Interpreter in Figure 30.

A summary of the general characteristics of the individual arrays is given in Table II. Appendix II is the final report from Texas Instruments Incorporated on the LSI arrays.

Texas Instruments informed Burroughs in December 1971 that they were discontinuing fabrication of LSI Discretionary Routed Arrays (DRA) after the conclusion of their present commitments. However, several alternative packaging approaches exist which could package the Interpreter logic as densely as in the LSI/DRA approach of Texas Instruments.

Figure 31. Aerospace Multiprocessor Installation at Wright-Patterson Air Force Base

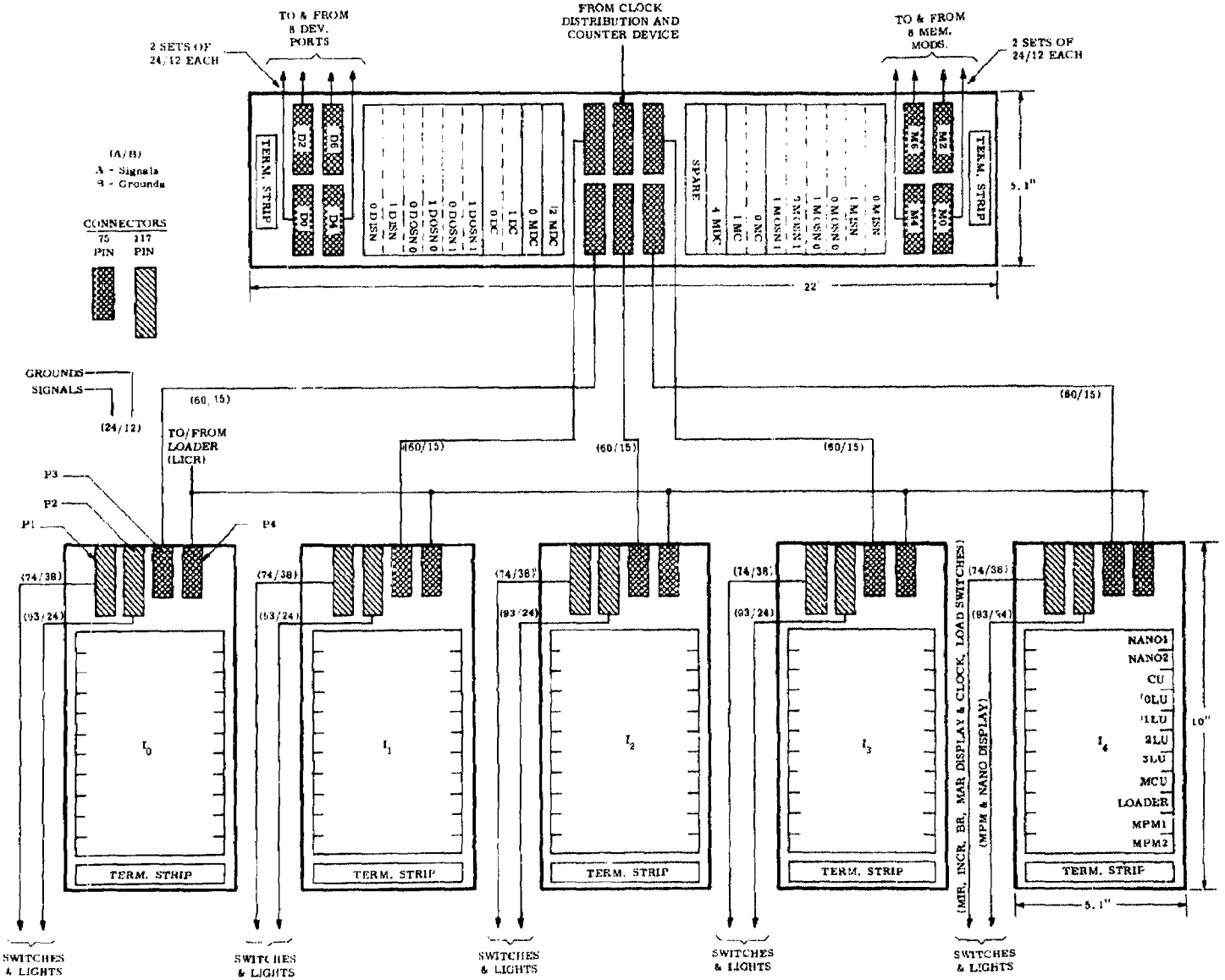


Figure 32. Multiprocessor Interconnection Scheme

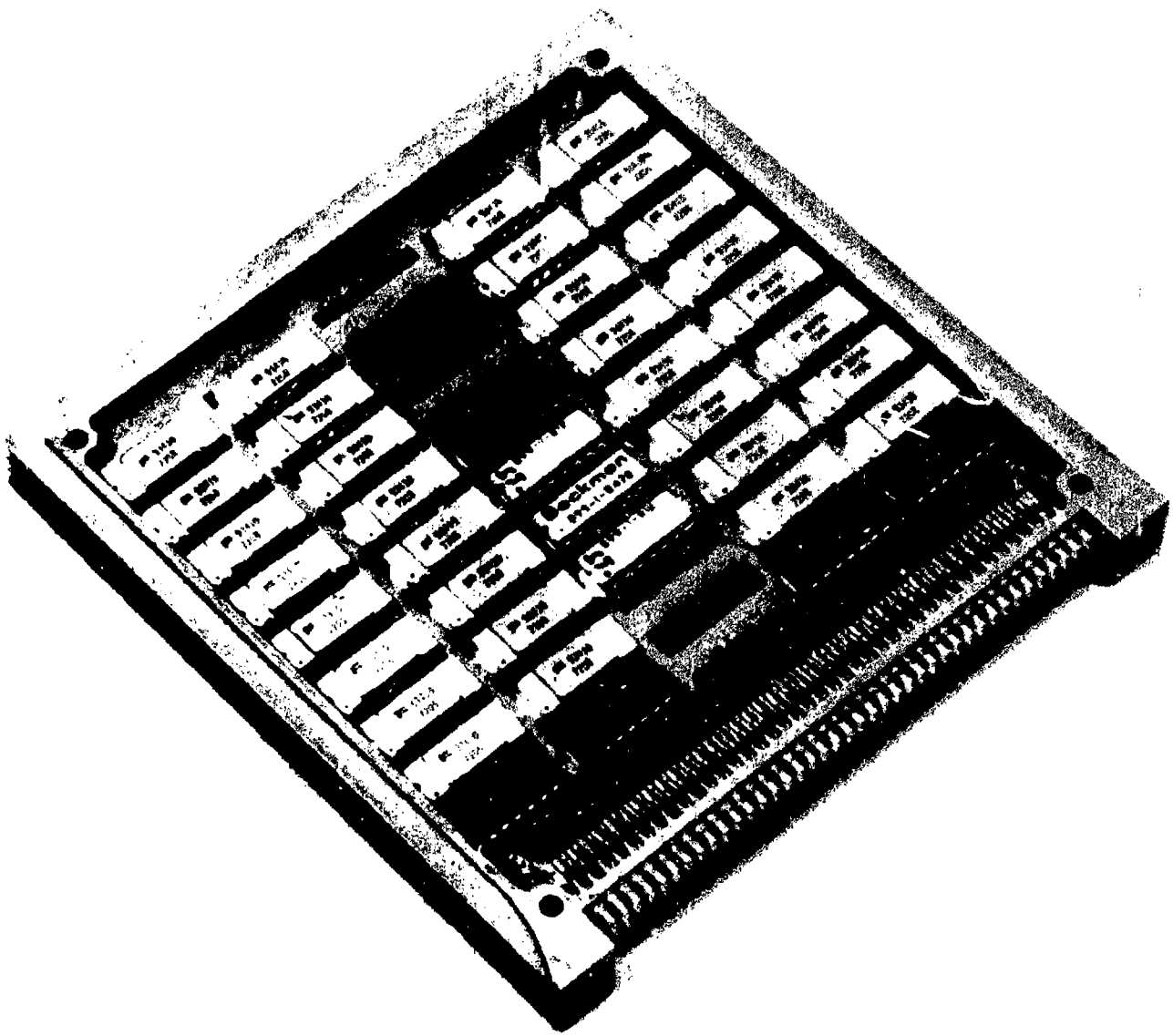


Figure 33. Microprogram Memory, Nanomemory Submodule Packaging

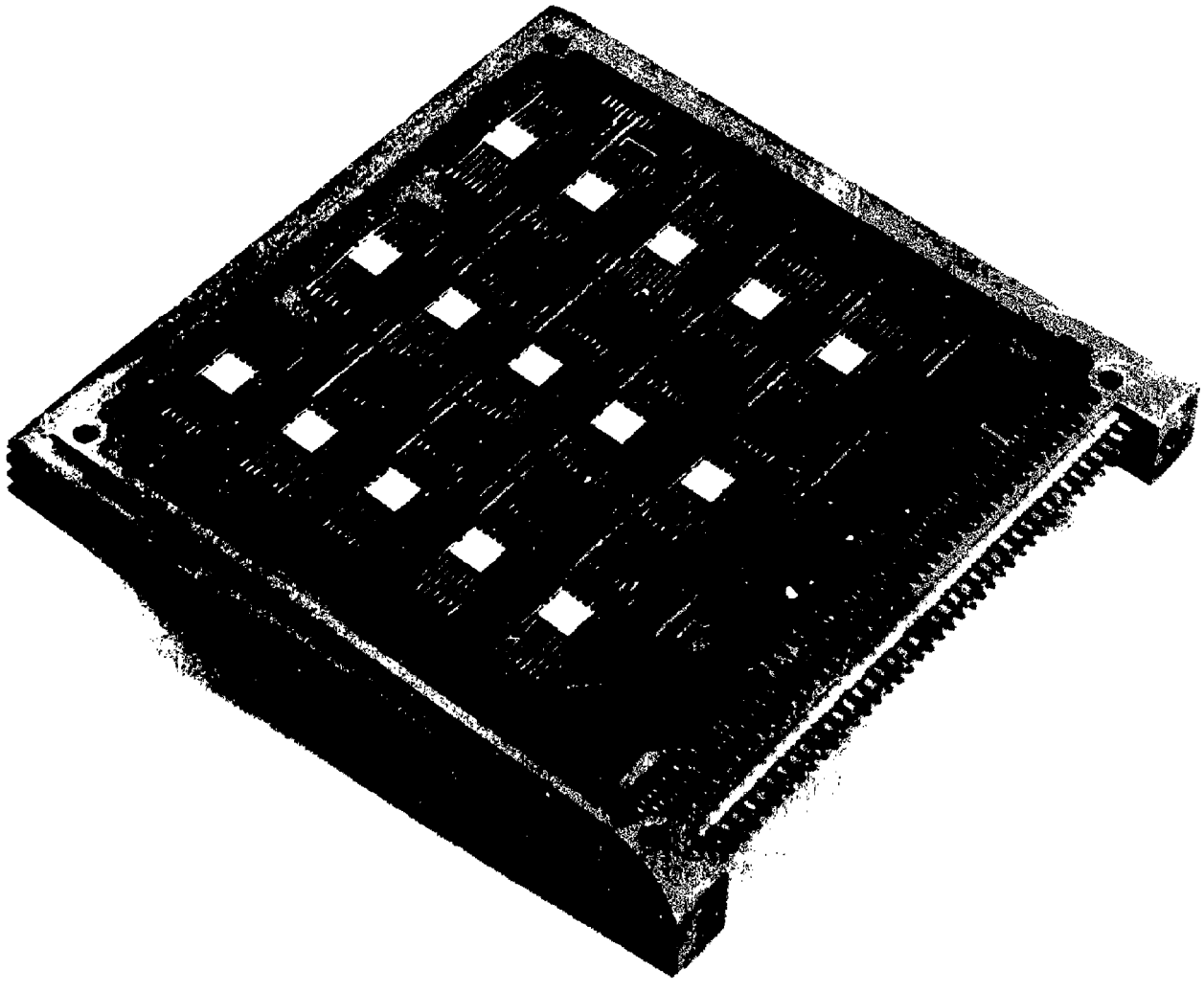


Figure 34. Loader, Switch Interlock Submodule Packaging

Table II. Multiprocessor LSI Array Details

Name	TI LSI/DRA Number	Function	Typical Power	Input Pins	Output Pins	Number of Cells					Equivalent Gates*	Number of Test Patterns Required
						Exclusive OR	3-inp. NAND	7-inp. NAND	ANI	FF		
LU #1	3013	B Reg., Adder, BSW1 Controls	3.14	67	26	18	93	15	30	8	420	338
LU #2	3014	BSW1, BSW2, A-Reg., MIR	4.10	47	34	16	113	17	26	32	562	207
CU #1	3015	SAR, Clock Controls, Adder Decode	2.82	36	35	9	83	0	21	22	389	452
CU #2	3016	Condition Reg., MPM Content Decode	2.71	40	15	10	68	17	25	14	374	786
MCU #1	3017	BR's, MAR, CTR, MPAD Controls	3.52	42	35	11	71	4	23	38	497	348
MCU #2	3018	MPCR, AMPCR, LIT, INCR.	3.97	55	34	9	86	16	31	36	562	531
N-slice		Total Available				80	232	56	82	100		
		(Recommended Usage)**				(18)	(70)	(17)	(25)	(30)		
S-slice		Total Available				18	96	30	46	58		
		(Recommended Usage)				(10)	(60)	(19)	(21)	(26)		

*Exclusive OR = 3 gates; 3-inp NAND = 1 gate; 7-inp. NAND = 1 gate; And-Nor-Invert = 7 gates; Flip-Flop = 6 gates

**Recommended Design with up to 30% of each single circuit type. This due to limitation on routing capability, not to circuit yield.

Three of the approaches are as follows:

1. A flat pack version of the multiprocessor can be produced with the same volume, weight and power requirements as the LSI version.

   The logic provided by two LSI chips can be duplicated with a maximum of 98, 16-pin flat packs as shown in Figure 35. With the use of multilayer boards, the 98 flat packs can be interconnected on the same 5-inch by 5-inch 1/2-inch thick heat sink as used for two LSI chips.

2. By utilizing 60-pin hybrid flat packs as produced by TI, it is possible to package two 8-bit Logic Units on a single heat sink as shown in Figure 36. The Control Unit and Memory Control Unit can be packaged together on a single heat sink to provide a reduction of 1/2 the original volume. This technique would use Shottky low-power TTL.

3. A third approach which would give the same volumetric density as the present LSI model would be to utilize Hughes LSI which is produced by a proprietary pad-relocation process. The Hughes chips could be produced as one for one replacement of the LSI arrays used in the present processor or as a replacement for the logic on two LSI arrays that are presently mounted on one of the submodular housings.

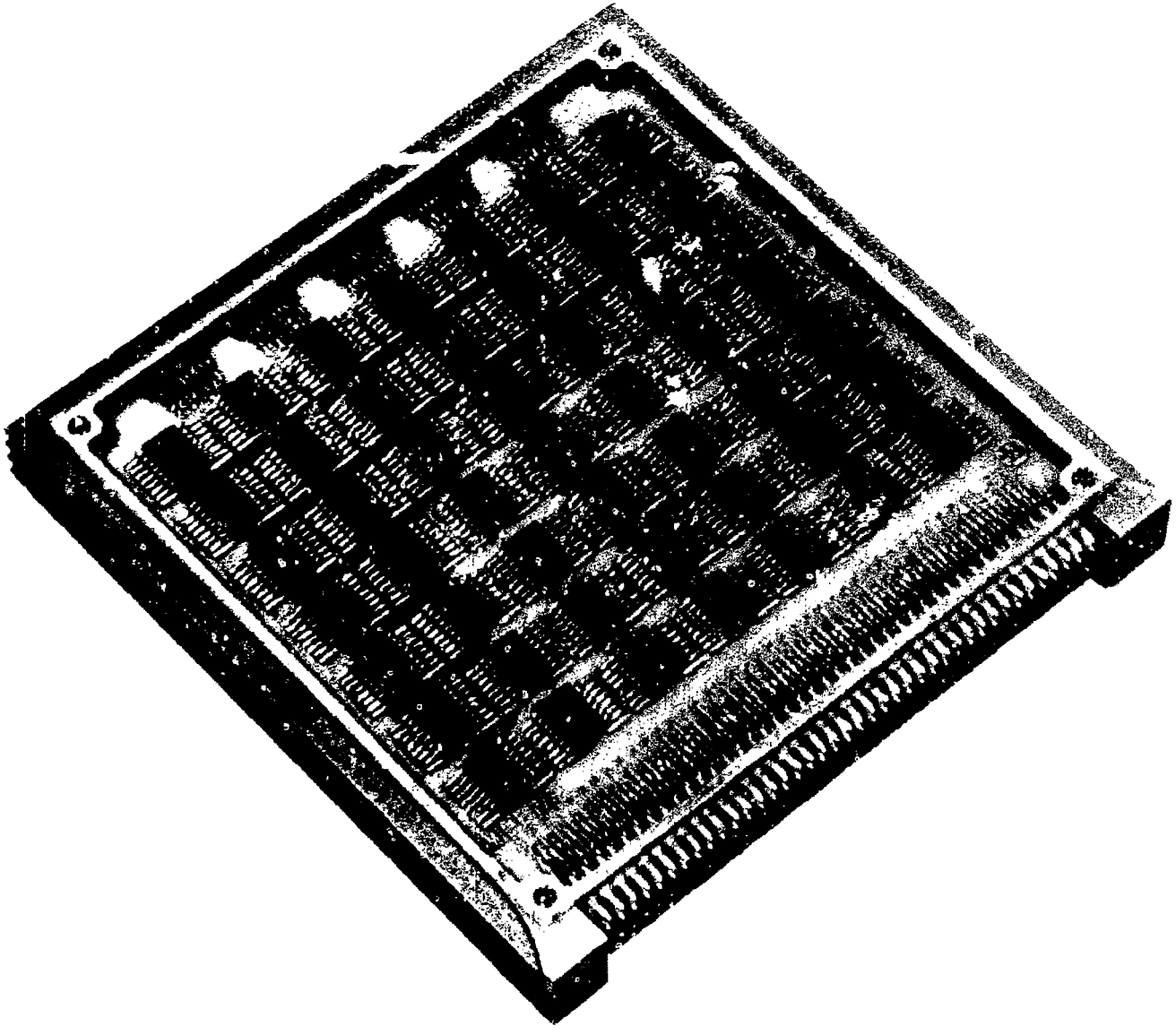


Figure 35. Alternative Packaging Approach Utilizing 16-pin Flat Packs

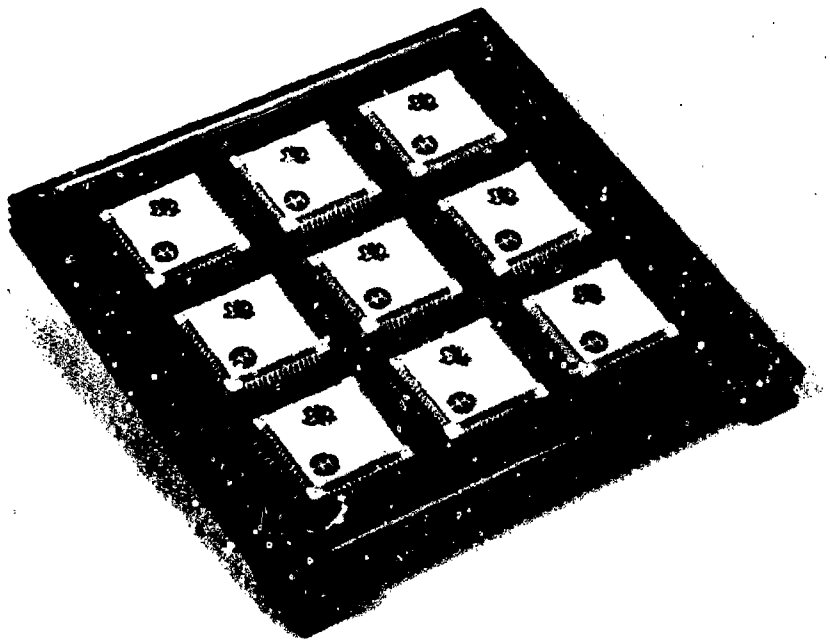


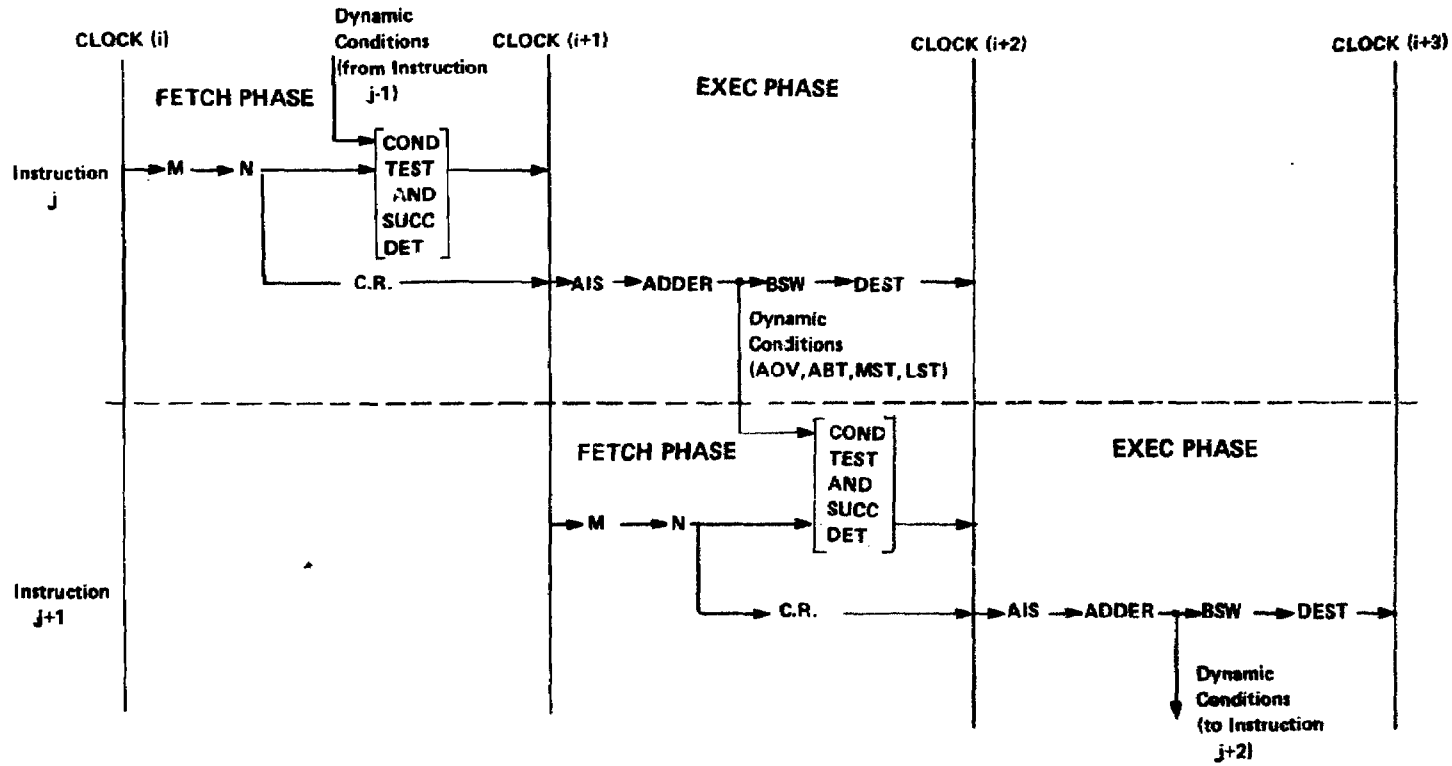
Figure 36. Alternative Packaging Approach Utilizing 60-pin Flat Packs

SECTION V

INTERPRETER OPERATION

During each clock period, a microinstruction is read from the MPM. The first four bits of this microinstruction indicate which of two types of instruction it is. If it is a Type I instruction, the remaining bits of the MPM word specify a Nanomemory address to be accessed. The Nanomemory is then initiated and its output, a set of 54 bits, provides the control functions as indicated in the listing below.

Nano-Bits	
1-4	Select a condition.
5	Selects true or complement of condition.
6	Specifies conditional or unconditional LU operation.
7	Specifies conditional or unconditional external operation (memory or device)
8-10	Specifies set/reset of condition.
11-16	Successor controls (wait, skip, step, etc.).
17-26	Selects A, B, and Z adder inputs
27	Byte carry control.
28-31	Selects Boolean and basic arithmetic operations.
32-33	Selects shift operation.
34-36	Enables input to A registers.
37-40	Selects input(s) to B register.
41	Enables input to MIR.
42	Enables input to AMPCR.
43-48	Enables and selects input to address registers and counter (MAR, BR1, BR2, CTR).
49-50	Selects input to SAR.
51-54	Selects external operations (read, write, lock, etc.), MPM load, or Nanomemory load.



M = MPM ACCESS TIME
N = NANO ACCESS TIME
COND TEST AND SUCC DET. = CONDITION TEST AND SUCCESSOR DETERMINATION
BSW = BARREL SWITCH
DEST = BARREL SWITCH OUTPUT DESTINATIONS;I.E., REGISTERS IB, CTR, ETC.) AND THEIR INPUT LOGIC
C.R. = CONTROL REGISTER AND ASSOCIATED LOGIC
AIS = ADDER INPUT SELECTION FROM COMMAND REGISTER

Figure 37. Timing Analysis, Type I Instructions

If the microinstruction is Type II, the remaining bits of the MPM word are stored into one or two registers: namely, the SAR, LIT, SAR and LIT, or the AMPCR. The determination of which registers are to be loaded is specified by the first bits of the MPM word. The Nanomemory is not accessed during a Type II operation.

Each Type I microinstruction has two parts (or phases). The first fetches the instruction from the MPM and Nanomemory and the second executes the fetched instruction. Figure 37 illustrates these two basic phases of each Type I microinstruction.

The fetch phase involves: MPM accessing, Nanomemory accessing, condition testing, selection of controls for the next instruction (successor) address computation, and, in parallel, loading the control register for the execution of the microinstruction. A fetch phase occurs for every Type I microinstruction and requires one clock time. Since it always overlaps the execution phase of a prior Type I microinstruction (Figure 37), the performance of each microinstruction requires effectively one clock interval.

The execution phase also requires one clock time and always overlaps the fetch phase of the next Type I instruction. The control signals for the execution phase are from the output of the control register and have two parts: signals specifying the logic unit operation (adder input selection, adder function, barrel switch shifting, etc.) and signals specifying the destination register(s) loading (i.e. clock enables). Both sets of these controls apply continuously from the start to the end of the phase; however, the destination registers are not changed until the occurrence of the clock pulse which signals the end of the execution phase and which simultaneously reloads the control register for the execution of a new logic unit operation. The completion of the execution phase (i.e. the destination register(s) loading), may be delayed or suspended for one or more clock times.

Suspended execution phase is the name given to an execution phase clock time whose logic unit operation has been and continues to be performed but whose destination register loading is postponed for one or more clock periods. This is accomplished by inhibiting clocks to both the control register and the destination registers. The register loading part of an execution phase depends on the subsequent microinstructions which follow the Type I instruction.

This suspended execution phase can occur for three primary reasons. The first and most frequent occurrence is when the next instruction from the MPM is a Type II instruction. This Type II instruction is executed during the same clock time it is fetched and the execution of the Type I instruction in progress is held in this suspended execution phase until the next clock interval. This allows the fetch phase of the next microinstruction (if it is a Type I) to have an execution phase to overlap. This provides condition bits (generated dynamically during the execution phase of a microinstruction) that can be tested during the fetch phase of the next Type I microinstruction.

A. Type I followed by Type I for which a logic operation is required:

1. Type I	F	E	
2. Type I		F	

B. Type I followed by Type II, followed by Type I for which a logic operation is required.

1. Type I	F	SE	E
2. Type II		II	
3. Type I			F

C. Type I followed by Type I for which no logic operation is required, followed by Type I for which a logic operation is required.

1. Type I	F	SE	E
2. Type I		F	
3. Type I			F

F Fetch
E Execution
SE Suspended Execution
} Type I

II Type II

Figure 38. Instruction Timing

This instruction overlap is more graphically illustrated in Figure 38 where the horizontal scale is "time". Example A of Figure 38 shows the case of sequential Type I instructions. Example B of Figure 38 shows the case of a Type I microinstruction followed by a Type II, which causes the execution phase of the preceding microinstruction (a Type I) to be suspended so that the execution will overlap the fetch phase of the third instruction (also a Type I). In case the third instruction had also been a Type II, the execution phase of the first microinstruction (the Type I) would have again been suspended. It is important to realize that since the execution phase of a Type I microinstruction is delayed by a Type II, the SAR, LIT, or AMPCR registers could be loaded with a value that would change the result of the operation during the completion of the execution of the Type I microinstruction.

The second reason for the occurrence of a suspended execution phase is due to the existence of conditional logic unit operations. A Type I microinstruction which does not contain a conditional logic operation always has a fetch phase and an execution phase. However, a Type I microinstruction which does contain a conditional logic operation falls into either of two categories: if the condition is met, both the fetch phase and execution phase will be performed; if the condition is not met, only the fetch phase will be done. However, even when the execution phase of a conditional Type I microinstruction is ignored, the fetch phase of the next Type I microinstruction must have an execution phase to overlap in order to have values for dynamic conditions. This is accomplished by forcing the prior Type I instruction into a suspended execution phase, which inhibits clocks from the destination registers and control register, which causes the execution phase of the current microinstruction to be disregarded. This is shown in example C of Figure 38. Example C shows a suspended execution phase occurring when the condition tested in the second microinstruction is not met, resulting in discarding the execution phase of that second instruction. More detailed examples explaining the above concepts appear in Figure 39, where CR refers to the command register, the vertical lines indicate the occurrence of a clock, and an X appears over clocks which are inhibited from occurring.

The other reason for a suspended execution phase is for use during the loading of the MPM and Nanomemory.

Since microprogram timing is important in the execution of microprograms on the Interpreter, the following summary of timing concepts must be kept in mind by the programmer in the creation of microprograms:

1. A fetch phase of a microinstruction is always executed in parallel with an execution (or suspended execution) phase of another microinstruction.

1. All Type I unconditional instructions

   a. A1 + B → A1

   b. A2 + B → A2

   c. A3 + B → A3

   d. A1 C → A1;

2. All Type I instructions where both AOV and ABT test true

   a. A1 + B → A1

   b. If AOV then A2 + B → A2

   c. If ABT then A3 + B → A3

   d. A1 C → A1;

3. All Type I instructions where AOV tests false; ABT tests true

   a. A1 + B → A1

   b. If AOV then A2 + B → A2

   c. If ABT then A3 + B → A3

   d. A1 C → A1;

4. Type I and Type II instructions Resulting A2 contains least 4 bits left justified

   a. 2 → SAR; 3 → LIT

   b. A2 and LIT C → A2

   c. 4 → SAR; 15 → LIT

   d. A1 C → A1;

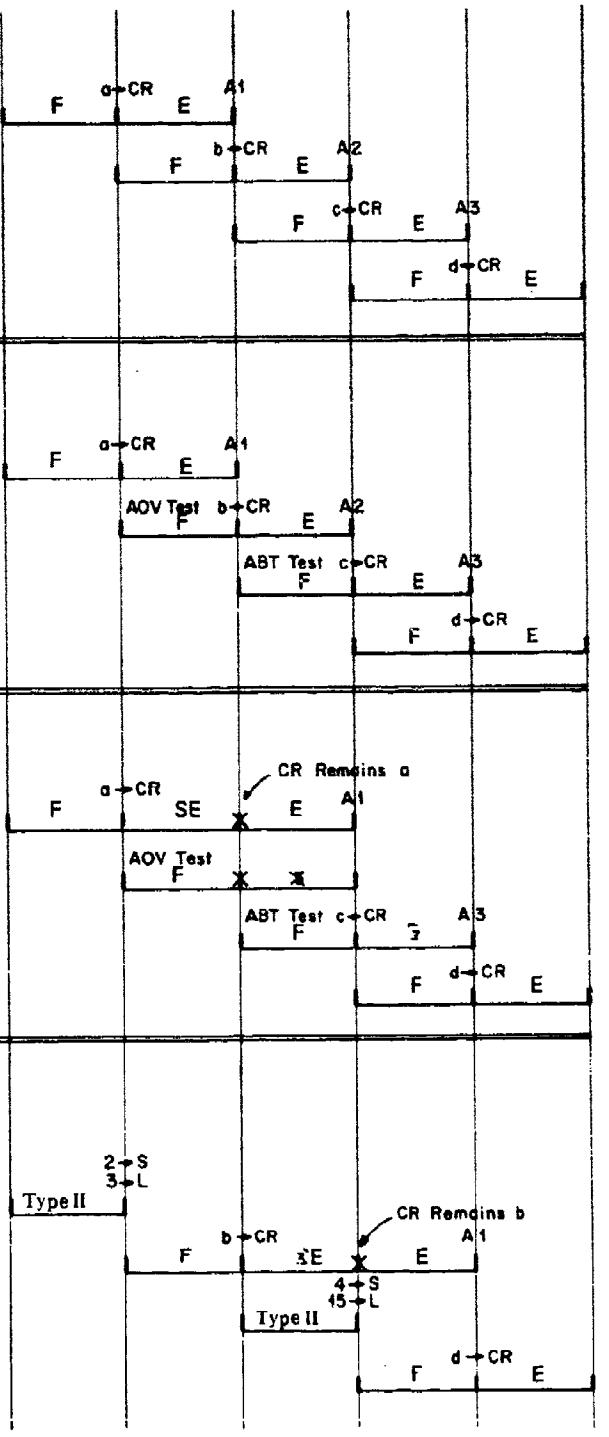


Figure 39. Timing Example

2. A suspended execution phase occurs primarily due to a successor that is either a Type II or a Type I microinstruction which contains a conditional logic unit operation that has not been satisfied.

3. A suspended execution phase of a Type I microinstruction which consists of both a fetch phase and an execution phase) does not become completed until the occurrence of another Type I microinstruction which also consists of both phases.

4. Any microinstruction which either causes a suspended execution phase to be initiated or prolongs an existing suspended execution phase is actually executed in time between the fetch phase and the execution phase of the affected Type I microinstruction although it may programmatically follow it.

The sequencing of microinstructions is also important in understanding the Interpreter operation.

The sequencing of Type I microprogram instructions is controlled by the following procedure: The MPM addresses the nanomemory which provides information to the condition testing logic indicating which condition is to be tested. The condition testing logic provides a True/False signal to the successor selection logic which selects between the three True and three False successor bits (also from the Nanomemory). The three selected bits (True/False) provide eight possible successor command combinations listed below and also shown in Figure 40. A Type II microinstruction (which does not access the Nanomemory) has an implicit STEP successor.

Wait	Repeat the current instruction
Step	Step to the next instruction
Skip	Skip the next instruction
Jump	Jump to another area of MPM (as specified by AMPCR)
Retn	Return from a Micro subroutine
Call	Call a Micro subroutine, saving the return address
Save	Save the address of the head of a loop
Exec	Execute one instruction out of sequence

The particular chosen successor command then provides controls used in the selection (MPCR/AMPCR) and incrementing logic which generates the next MPM address. Except for the EXEC command, the MPCR is loaded with this MPM address.

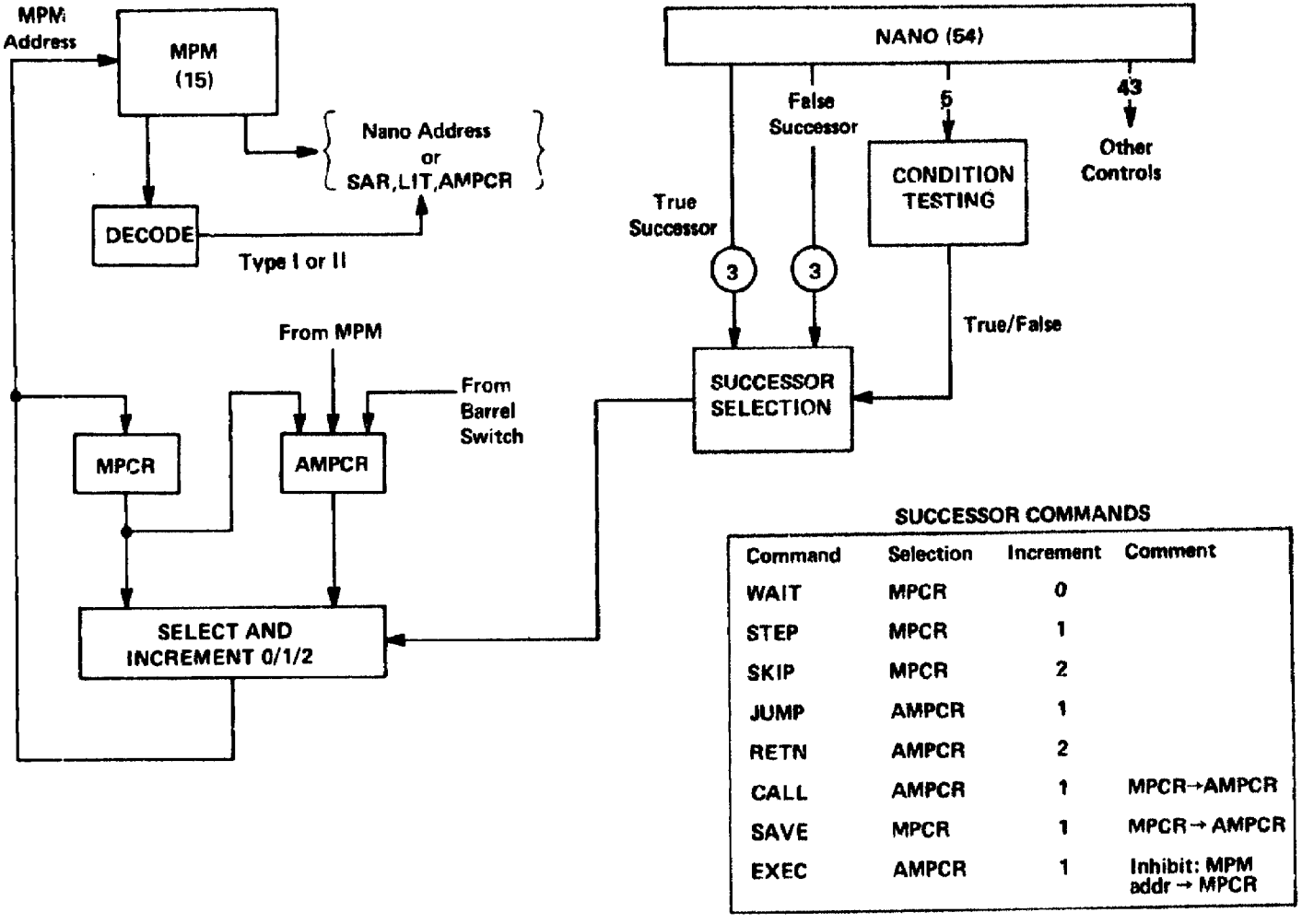


Figure 40. Microprogram Instruction Sequencing

SECTION VI

SWITCH INTERLOCK OPERATION

OVERALL SWITCH INTERLOCK CONTROL AND TIMING

Figure 41 is a block diagram of the Switch Interlock (SWI), connecting five Interpreters to eight devices and eight memory modules. The transmission paths through the SWI break the 32-bit data word into 8 wires carrying 4 serial bits each.

Only Interpreters can issue control signals to access memories or devices. A memory module or device cannot initiate a path through the Switch Interlock, but it may, however, provide a signal to the Interpreter to an unused condition bit via a display register*, a device connected to the SWI. Thus, transfer between devices and memories must be via and under the control of an Interpreter. Connection with a device-like port is by "reservation" for exclusive use by an Interpreter and is maintained until released by that Interpreter or in the case of that Interpreter failing. (A memory could be attached to a device-like port if locking of an Interpreter to a memory is desired.) Connection with a memory-like port is for the duration of a single data word exchange. (Note also that a device could be attached to a memory-like port. To simplify the description however, these two types of ports will be referred to just as device ports and memory ports in the following discussion).

*No display register is being delivered with the aerospace multiprocessor, but is an easily designed device that could take a variety of forms. Basically, setting any bit in the display register would set the condition bit in the Interpreter. When this bit is tested true, the display register would be read, returning either the entire register, a masked portion of the register, or possibly the address of the device with the highest priority interrupt, depending upon the design of the display register device.

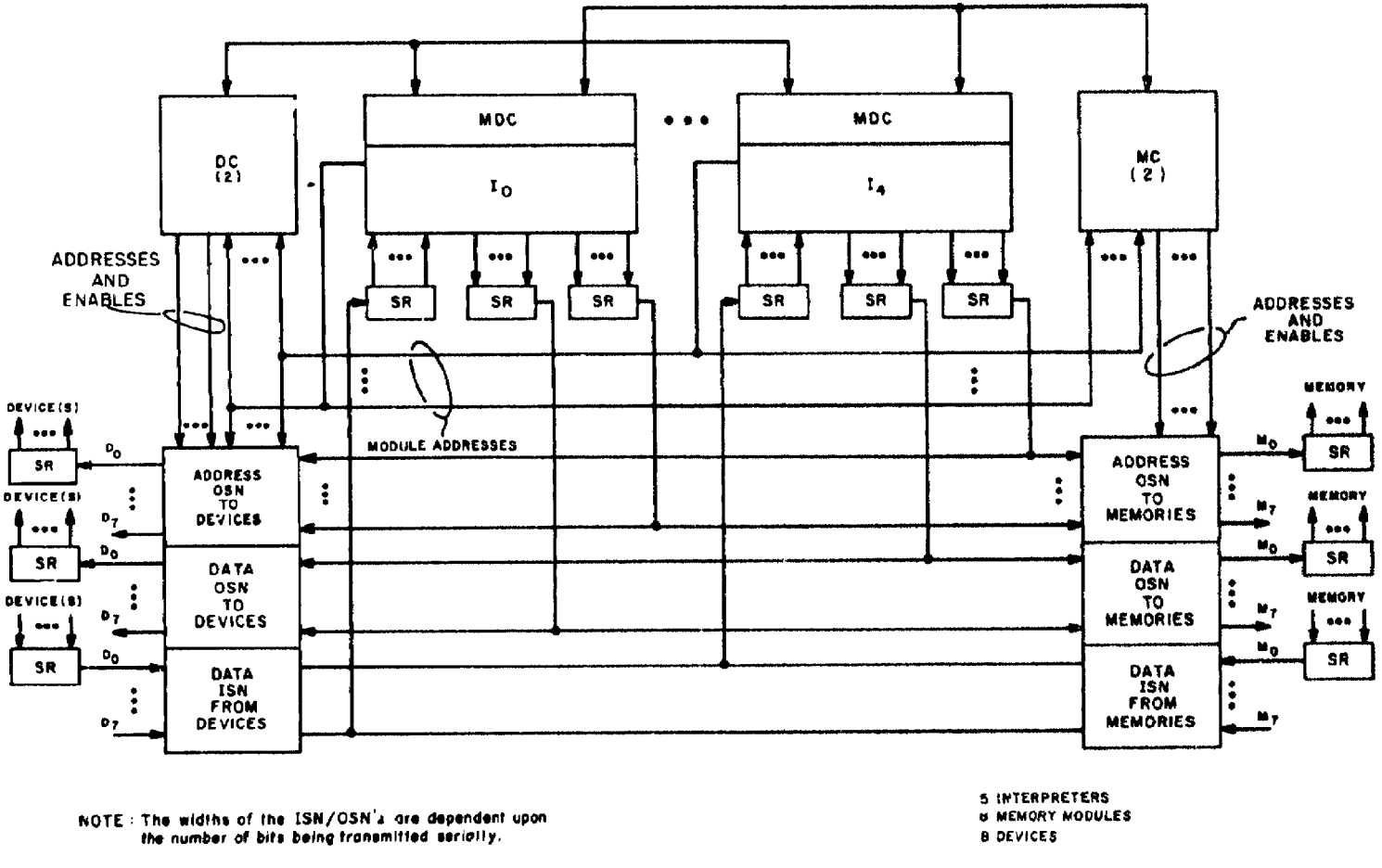


Figure 41. Switch Interlock, Block Diagram

Controls are routed from the Interpreters through the MDC to the MC and the DC which, in turn, check availability, resolve conflicts, and perform the other functions that are characteristic of the Switch Interlock. Data and addresses do not pass through the MDC, but are wired directly to the OSN's.

Events are initiated by the Interpreter for access to memories or devices. The Interpreter awaits return signals from the MDC. Upon receipt of these signals, it proceeds with its program. Lacking such positive return signals, it will either wait, or retry continuously, depending upon the Interpreter program (and not on the Switch Interlock). A timeout waiting for a response will be performed by a counter (called the "HORN") that will force a STEP in the microprogram after a preset length of time and will set a condition bit to indicate a failed memory module or device due to the lack of a response. This counter is reset every time any memory or device operation is done.

Among the significant signals which are meaningful responses to an Interpreter and testable as conditions are the following:

Switch Interlock has Accepted Information (SAI)	The MAR and MIR of the Interpreter may be reloaded and a memory or device has been connected.
Read Complete (RDC) or Request of Device Complete (RDC)	Data is available to be gated into the B register of the Interpreter or the device written to has accepted its information.
Horn Overflow (HOV)	No memory or device operations have have been performed for the last 2^{20} Interpreter clock times.

The rationale for this "handshaking" approach is consistent with the overall Interpreter-based system design which permits the maximum latitude in the selection of memory and device speeds. Thus the microprogrammer has the ability (as well as the responsibility) to provide the timing constraints for any system configuration.

For each Interpreter, the Switch Interlock provides three buffer shift registers.

1. Address data for S memory and devices from the specified MAR1 or MAR2. (XDA).

2. Output data from the MIR. (XDO).

3. Input data for assembly and subsequent acceptance into the B register. (XDI). Data in this register may be repeatedly read non-destructively until the next device or memory operation is initiated (the last read may be concurrent with the next operation), provided no intervening instruction uses a B register input selection involving the MIR.

DEVICE OPERATIONS

The philosophy of device operations is based upon an Interpreter using a device for a "long" period of time without interruption. This is accomplished by "locking" an Interpreter to a device. (The reader is reminded that a memory could be attached to a "device-like" port.)

The device operations include lock (DL), read (DR), write (DW), and unlock (DU). Each device operation uses as a device identification the value of the most significant three bits of BR1 or BR2 as indicated in the operation suffix, e.g., DL1. This identification is not stored by the Switch Interlock; consequently it must be maintained until the device operation is completed, or until some other device or memory action is desired. Any change to the device identification while a device operation is in progress breaks the selected path to or from the Interpreter. Unless the normal completion occurs concurrently, the prior device operation is terminated. The value in MAR and in the least significant 6 bits of BR1 or BR2 pass through the Switch Interlock to the device as required. A signal indicating read or write is placed in the most significant bit of the XDA shift register in place of one of the module address bits which are not needed by the memory module or device.

The ground-rules for device operations are listed below:

1. An Interpreter must be locked to a device in order to read from or write to that device.

2. An Interpreter may be locked to several devices at the same time.

3. A device can only be locked to one Interpreter at a time.

4. When an Interpreter is finished using a device, it should be unlocked so other Interpreters can use it. Devices locked to to a failed Interpreter are unlocked by turning power off to the failed Interpreter.

A block diagram of the DC is given in Figure 18 in the Multiprocessor Hardware Section of this report. One primary purpose of the DC is to resolve conflicts in device lock (DL) and device unlock (DU) requests that may occur.

The second purpose of the DC is to check to make sure a device is locked to an Interpreter that is requesting to read from, write to, or unlock from that device. This is accomplished by the "Lock Check for Device Operation" in the right of Figure 18.

If an Interpreter issues a read or write command in an attempt to control a device, and it has not previously locked the device, it will not be given access to the device regardless of its (the Interpreter's) priority status. However, as stated above, if it had previously locked the device, it has explicit priority to that same device.

Device Lock and Unlock

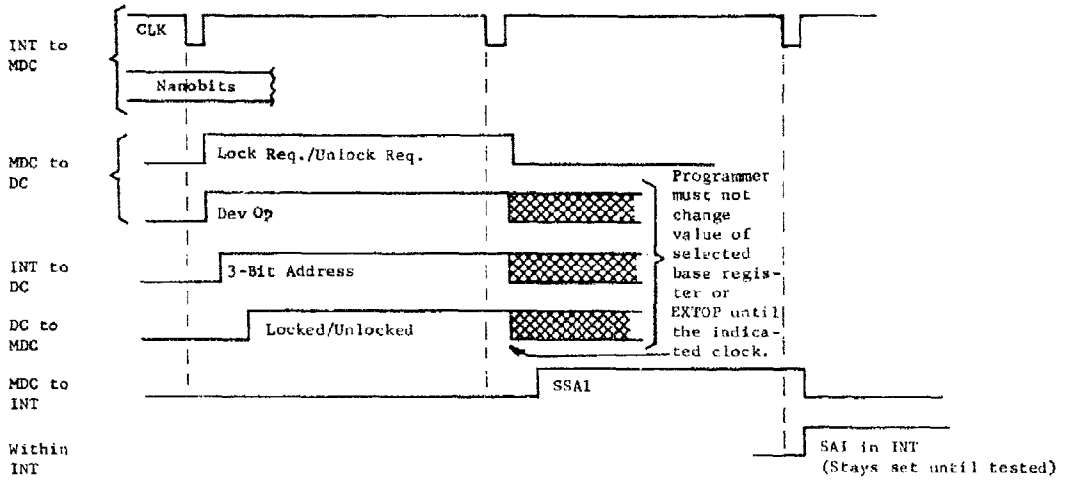
Timing diagrams for DL and DU operations are shown in Figures 42 and 43. In both cases, controls from the Interpreter (Nanobits 51-54) are strobed into the mem/dev operation register of the MDC if either the Type I microinstruction is unconditional or the selected condition is true, independent of whether the next instruction is Type I or Type II. A Device Operation signal and either a Lock Request or an Unlock Request are derived from the output of this register and are sent from the MDC to the DC, concurrent with a 3-bit address being sent to the DC from the selected base register output of the Interpreter.

For the case of either a DL to a device previously locked to the requesting Interpreter or a DU to a device previously unlocked from any Interpreter (shown in Figure 42), an appropriate status signal is returned from the DC to the MDC, and conflict resolution for actually performing the DL or DU is of no consequence. In these two cases, the flip-flop in the MDC for synchronizing the SAI signal is set with the next clock. The actual SAI flip-flop in the Interpreter will then be set with the second clock and will test true during the fetch phase of the third instruction following the DL or DU.

However, for the cases of a DL to an unlocked device or a DU to a device locked to the requesting Interpreter (shown in Figure 43), conflict resolution is necessary. The DL request from the highest priority requesting Interpreter is honored over a co-occurring request for the same device from any lower priority Interpreter. Concurrent DL or DU requests for different devices may cause the lower priority request to incur a one clock delay in achieving the DL or DU and in return of SAI, for each higher priority request. Consequently DL or DU requests from Interpreters other than the highest priority may be arbitrarily delayed. The earliest confirming SAI response occurs 3 instructions after issue of the DL or DU. If SAI is true, then the DL or DU was successful. If SAI is false, then it means that the DL or DU is not yet successful. The design justification for this potential arbitrary delay is that DL or DU are infrequent events for which arbitrary delay is of little consequence.

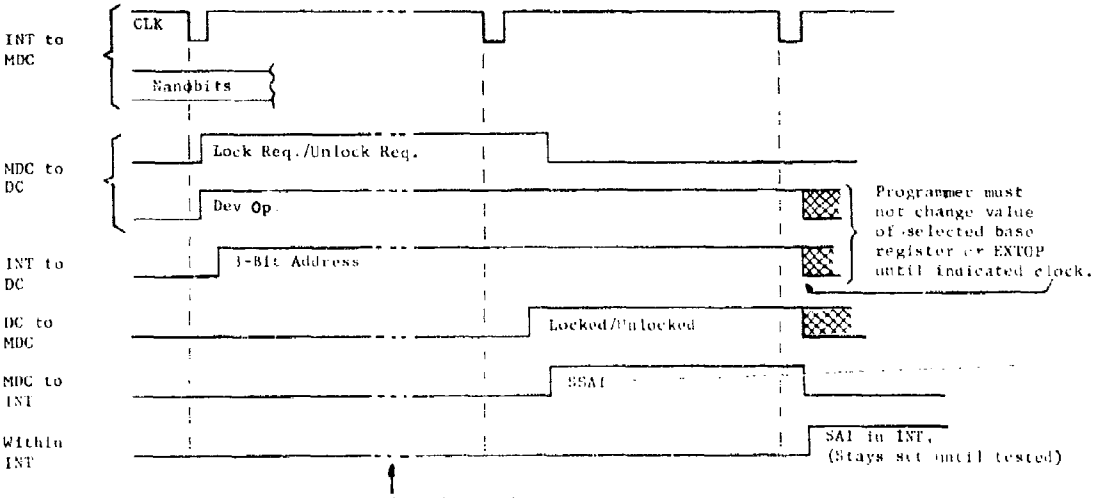
Device Read and Write

A timing diagram for DR or DW is shown in Figure 44. As for DL and DU, controls from the Interpreter (Nanobits 51-54) are strobed into the mem/dev operation register of the MDC if either the Type I microinstruction is unconditional or the selected condition is true, independent of whether the next instruction is Type I or Type II. Controls derived from the output of this register will next load the output shift registers of the Interpreter and will send a Device Operation signal



NOTE: Levels on diagram indicate validity or nonvalidity and not necessarily logic level.

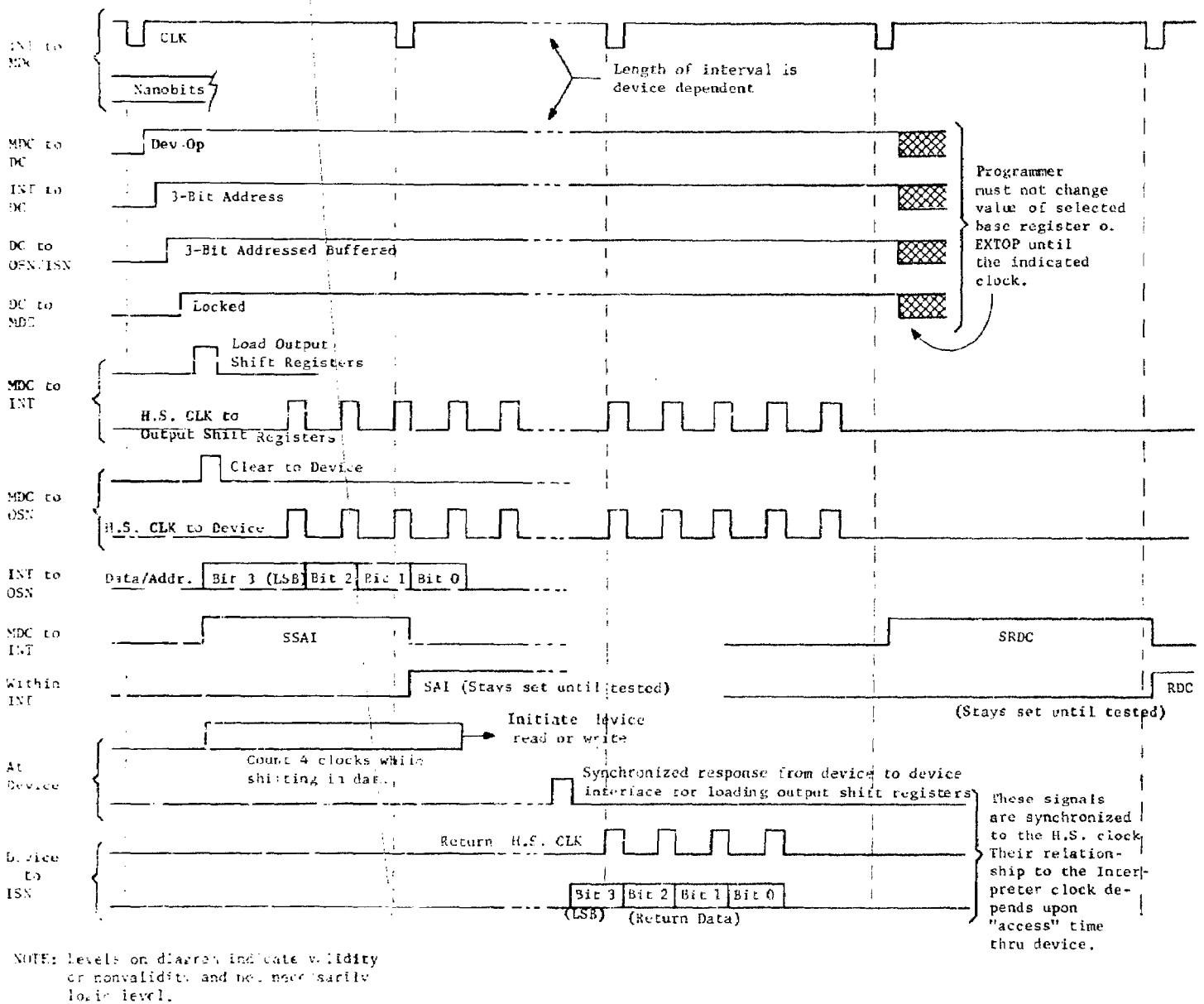
Figure 42. Timing Diagram for Device Lock to Device Previously Locked to Requesting Interpreter or for Device Unlock to Device Previously Unlocked from Any Interpreter



NOTE:
Levels on diagram indicate validity or nonvalidity and not necessarily logic level.

Interval of integral number of clock periods. Exists if higher priority interpreter(s) requesting lock to or unlock from ANY device. Interval = 0 if no higher priority requests.

Figure 43. Timing Diagram for Device Lock to Unlocked Device/ Unlock to Device Locked to Requesting Interpreter



NOTE: Levels on diagram indicate validity or nonvalidity and not necessarily logic level.

Figure 44. Timing Diagram for Device Read or Write from Device Locked to Requesting Interpreter

from the MDC to the DC, concurrent with a 3-bit address being sent from the selected base register of the Interpreter to check the lock status of that device. After it is confirmed that the device is locked, the DC returns a signal to the MDC which will cause a clear pulse to be sent to the device interface logic through the device OSN and will initiate the setting of SAI and the transmission of high speed clocks to the output shift register of the Interpreter and through the OSN's to the device interface.

For both a DR and DW, the device interface counts four clocks coming into it and then stops accepting high speed clocks. In the case of a read, the device interface usually waits for some kind of Data Available signal* from the device (such as "column strobe" from a card reader) which it will use to load its output shift registers and to allow four high speed clocks which are still arriving from the OSN to clock these output shift registers and to be returned to the MDC and the Interpreter with the serial data. The MDC will count four return clocks and will set a flip-flop in the MDC for synchronizing RDC. This signal is sent from the MDC to that Interpreter, for setting RDC, which then will test true during the following clock time. The value in the selected base register must not be changed during a device read, as shown in the timing diagram.

In the case of a write, the response is very dependent upon the particular device being interfaced. For the card reader, the next four high speed clocks are turned around and sent back to the Interpreter (status was chosen to be sent back as a "bonus"). In the case of the printer, a signal saying the last character was accepted by the printer is used by the device interface to allow return clocks. The four return clocks are counted by the MDC and are used as a means of saying that the device accepted the data sent out by setting RDC as for a DR. As in the case of a device read, the value in the selected base register must not be changed during a device write.

Device Use Sequence

The sequence of device operations necessary for an Interpreter to use a device is as follows:

1. A test of IF SAI should be included in some instruction to reset it. This usually can be in the instruction with the unconditional device operation.

2. Device Lock Request: The most significant three bits of the indicated base register are used as the device identification. The third following clock time will be the earliest SAI could have become true. SAI is then tested.

*Devices such as the real time clock (described in the Multiprocessor Hardware section) however, do not require a signal such as Data Available for synchronization since they are already synchronized to the Interpreter clock.

2.1 If true, then the device lock was successful.

2.2 If false, then the device lock was unsuccessful. The request remains in progress while other instructions not changing the device identification or issuing other memory or device operations may be executed. The DL request is terminated by the first of the following actions:

(a) The Interpreter initiates another memory or device operation.

(b) The Interpreter changes the device identification in the selected base register.

(c) The device becomes available and sets SAI. All co-occurring actions are valid. Should (a) and (c) co-occur or (b) and (c) co-occur, SAI refers to the DL for the following two instruction times and should be tested. In the instructions thereafter, SAI refers to the new memory or device operation. Should termination by (b) occur without co-occurrence of (c), the new device identification applies to the DL still in progress, and the path for SAI return is diverted to the newly identified device (if there is one so identified) without reissue of another DL.

3. Once the desired device is locked to the Interpreter, a sequence of one or more data exchanges may be initiated using a device write or device read.

4. Device Write: The data in the indicated base register is used to specify the device, and the data in the MIR provides the information to be written to the device. The second instruction after the device write, SAI may be tested. If true, the Interpreter is locked to the device, the data in the MIR has been accepted by the XDO register, and so the MIR may subsequently be changed. If false, the Interpreter was not locked to the requesting device.

The device provides four high-speed return clocks to generate an RDC when it has completed the requested write. Similar to DL, the request continues until the first of the corresponding 3 actions.

(a) The Interpreter initiates another memory or device operation.

(b) The Interpreter changes the device identification.

(c) The DW is completed and sets RDC. All co-occurring actions are valid. Should (a) and (c) co-occur or (b) and (c) co-occur, SAI refers to the DW for the following instruction time and should be tested. In the next following instruction SAI then refers to the new memory or device operation. Should (b) not co-occur with (c), then the DW in progress is diverted to apply to the new device identification without reissue of another DL.

5. Device Read: The data in the specified base register is used to specify the device. The second instruction after the device read, SAI may be tested. If true, the Interpreter is locked to the device; otherwise not.

   The device provides four high speed return clocks with the returning data to generate an RDC after the device read. Thus, the same instruction that finds RDC true may include BEX. RDC should be reset by testing prior to use for device read (usually as part of the prior instruction using BEX).

6. Device Unlock: When use of the device is completed, the lock should be terminated by issuing a device unlock. An SAI is returned if the issuing Interpreter was locked to the device. An attempt to unlock a device that is not locked to the Interpreter will not return SAI. SAI is available for test at earliest the third instruction after the device unlock.

MEMORY OPERATIONS

Memory modules normally cannot be locked and are assumed to require minimum access time and a short "hold" time by any single Interpreter. (The reader is reminded that a device could be attached to a "memory-like" port.) Conflicts in access to the same module are resolved in favor of the highest priority requesting Interpreter. Once access is granted, it continues until that memory operation is complete. When one access is complete, the highest priority request is honored from those Interpreters then in contention.

The memory operations include read (MR) and write (MW). Each memory operation uses as a memory address the value in MAR1 and MAR2 (BR1 or BR2 concatenated with MAR). The most significant 3 bits of the address specifies a memory module with the rest indicating locations within the module.

The MC, shown in Figures 19 and 20 of the Multiprocessor Hardware section of this report, provides for resolution of conflicts (this is fixed or wired priority) among contending Interpreters. Once conflicts have been resolved and access has been granted to a memory module by an Interpreter, the MC "remembers" this connection throughout the memory operation, allowing the selected base register to be changed as opposed to requiring the selected base register value to be maintained as for device operations. This register also allows for future

modification to the MC to allow "remembering" the connection until that Interpreter uses a different memory module. This would allow almost a one clock time faster access to the memory module if the next request is also to the remembered memory module, since no priority resolution need take place. More specifically, when a memory module would be requested by an Interpreter, the module name would be compared with the register which would contain the number of the last module which that Interpreter accessed. If it would match, the priority logic would then be bypassed, thus saving time. If it would not, it would mean that the memory either had been previously used by another Interpreter, or would presently be in contention for by other Interpreters, or would presently be in use by another Interpreter. In this case the requesting Interpreter would route its request through the priority logic (a few gate levels of delay). When access would be granted, the memory module address would then be clocked into the register in the part of the MC for the requesting Interpreter by the next Interpreter clock and the register for any other Interpreter containing that address would be reset to all zeros.

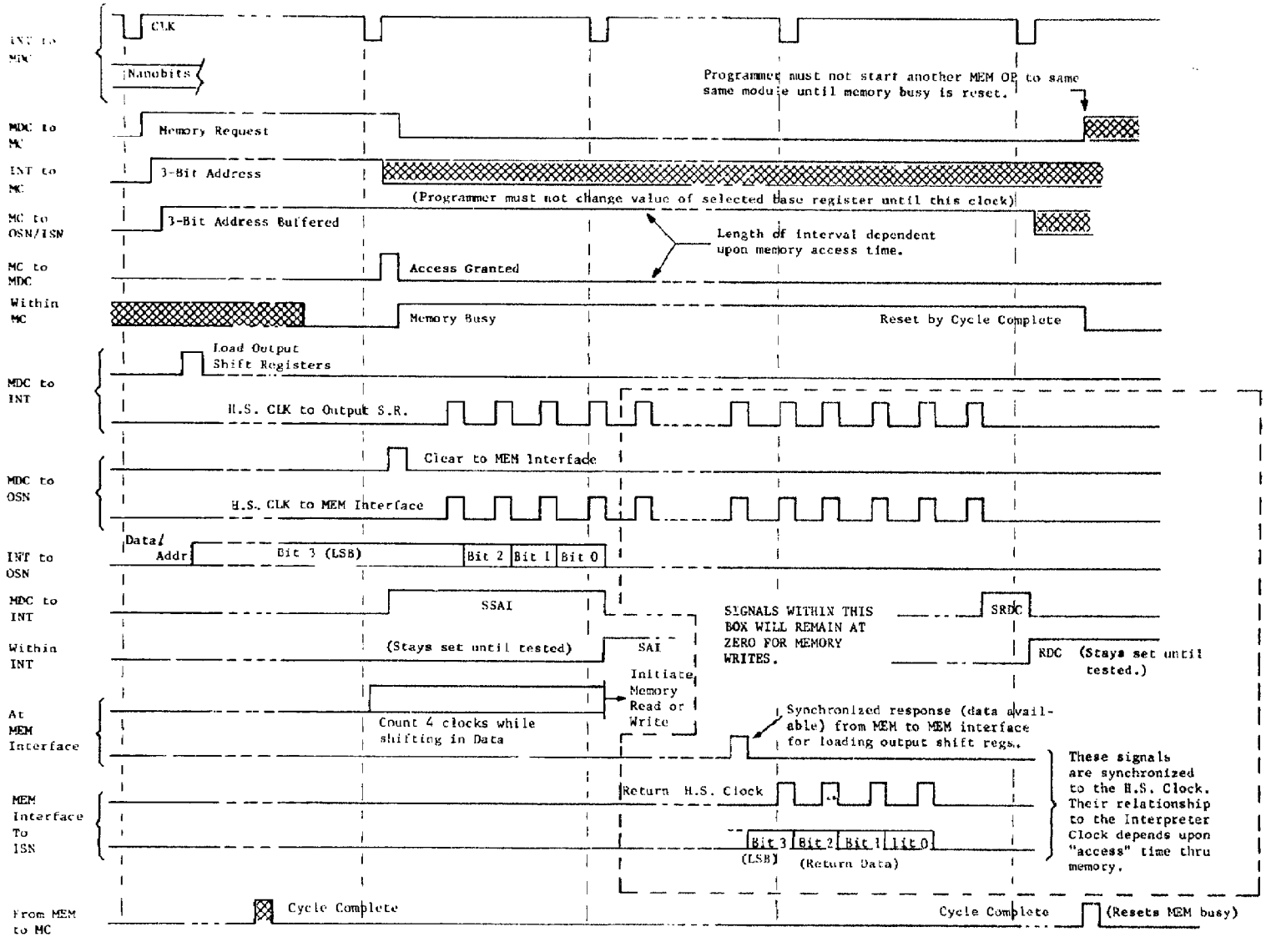
If locking of a memory module is required for purposes of block transfers or similar reasons, a memory is designated as a device and is placed under the control of the DC in which locking is permitted.

Memory Read and Write

A timing diagram for MR and MW is shown in Figure 45. As for device operations, controls from the Interpreter (Nanobits 51-54) are strobed into the mem/dev operation register of the MDC if either the Type I microinstruction is unconditional or the selected condition is true, independent of whether the next instruction is Type I or Type II. Controls derived from the output of this register will next load the output shift registers of the Interpreter and will send a Memory Request signal from the MDC to the MC, concurrent with a three bit address being sent from the selected base register of the Interpreter. This initiates the priority logic in the MC. When the MC has granted access by that Interpreter to the memory module it was requesting, a signal is returned from the MC to the MDC that will cause a clear pulse to be sent to the memory interface logic through the memory OSN and will initiate the setting of SAI and the transmission of high speed clocks to the output shift registers of the Interpreter and through the OSN's to the memory interface.

In the case of a memory write, the counter in the MDC will count four output high speed clocks and will then stop them.

In the case of a memory read, output high speed clocks are not counted. Instead, these high speed clocks are continually sent to the memory module interface. This interface will count four clocks coming into it and will then initiate a memory read. Upon return of a data available signal from the memory, the memory interface will load its output shift registers and then allow four of the high speed clocks that are still coming through the OSN to clock these output shift registers and to be returned to the MDC and the Interpreter with the shifted



NOTE: Levels on diagram indicate validity or nonvalidity and not necessarily logic level.

Figure 45. Timing Diagram for Memory Read or Write

out data. The MDC will count four of these memory return clocks and will then stop the high speed output clocks and set RDC indicating that the data has been shifted into the Interpreter input shift registers and is ready to be strobed into the B register.

Memory Use Sequence

The sequence of operations necessary to access S memory is simple in single Interpreter systems where no conflict in access can exist. In such cases once the address setup is complete (as is the MIR for write), the memory read (or write) can be initiated. After a suitable time the data from memory can be accessed via BEX or BBE. In the presence of conflict potential, the following control sequence should be used. This sequence is recommended for systems without a Switch Interlock as well.

1. Memory read

   1.1 A test of RDC should be included in some prior instruction in order to reset RDC. By convention this should be the previous memory read (or device read or write). A test of SAI also should be included in some prior instruction in order to reset SAI if address register changes are required after issuing the memory read before the RDC is returned, or if confirmation of access to the switch interlock is desired.

   1.2 The address should be in the selected base register and MAR.

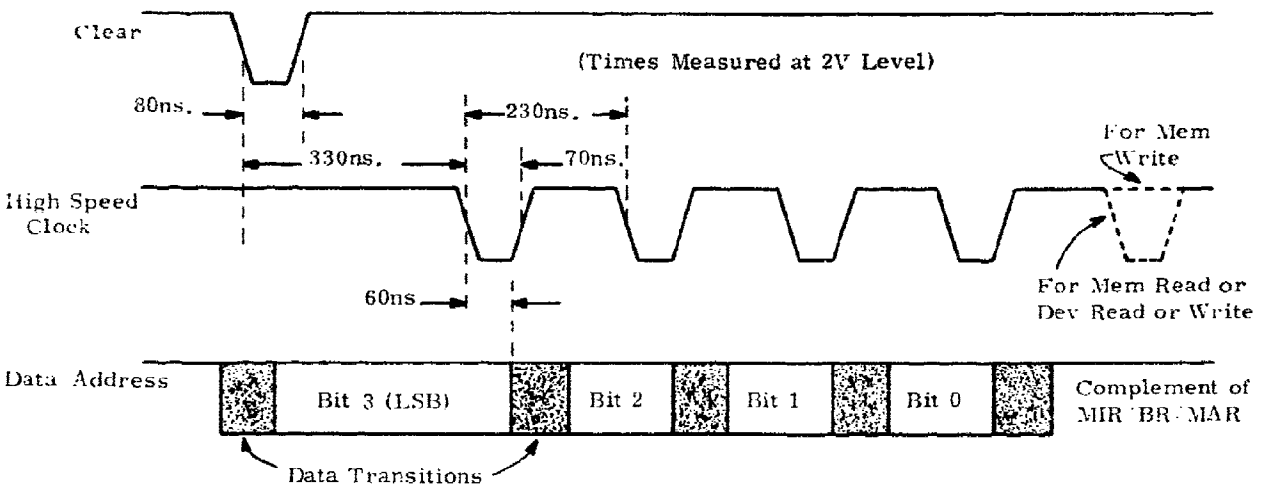
   1.3 The memory read can then be initiated the instruction after the address has the desired value.

   1.4 An SAI is returned when the Switch Interlock has accepted the address and the memory is connected to the requesting Interpreter through the Switch Interlock.

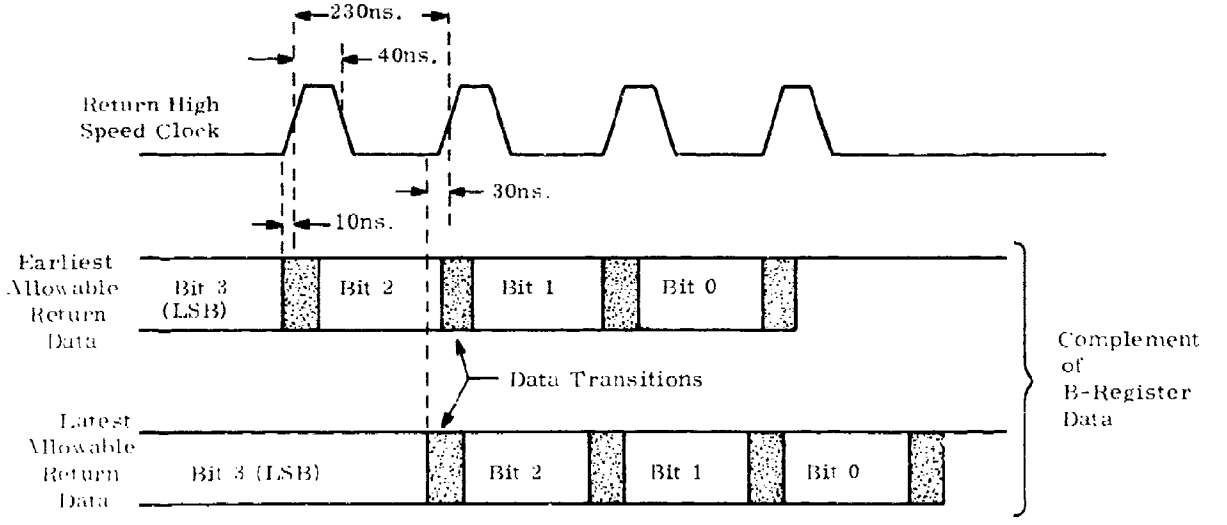
   1.5 A group of intervening instructions can be issued, depending on the relative speeds of the Interpreter clock and the S memory. Once SAI is set and tested, these instructions may change the address registers.

   1.6 An RDC (read complete) signal is returned when data is available for entry into the Interpreter.

   1.7 If no intervening device or memory reads occur and no intervening instruction used a B register input selection involving the MIR, BEX may be repeated, each time receiving the data in XDI non-destructively.



(a) Timing of Signals from SWI to Interface



(b) Timing of Signals to SWI from Interface

Figure 46. SWI/Interface Timing Signals

2. Memory Write

2.1 A test of SAI should be included in some prior instruction in order to reset SAI.

2.2 The data to be written should be in MIR.

2.3 The address should be in the selected base register and MAR.

2.4 The memory write can then be initiated the instruction after both the address and data have the desired values.

2.5 Return of SAI indicates that the memory is connected and therefore the address and data have been accepted in the XDA and XDO buffer registers respectively, and thus the address registers and MIR may be subsequently changed.

INTERFACE TO SWI

The interface to each memory or device port is functionally identical. For the aerospace multiprocessor, the interface from the SWI to the memory or device interface consists of a clear line, a high speed clock line, 8 data lines of 4 serial bits each and 4 address lines of 4 serial bits each. (The most significant bit of the BR is replaced by a read/write signal in the serial address sent to the memory or device port.) The interface from the memory or device interface to the SWI consists of a return high speed clock line and 8 data lines of 4 serial bits each.

The relative timing of these signals at the interface is shown in Figure 46. The timing in this figure was measured using one Interpreter and memory module only at the indicated frequency and should not be interpreted as resulting from any worst case timing analysis. In Figure 46a, the 330 nanosecond delay from clear to the high speed clock becomes smaller as the frequency of the high speed clock is increased. The widths of the clear and the 60 nanosecond delay from high speed clock to data are independent of the frequency or width of the high speed clock. In Figure 46b, the relationship between data and clock should be independent of the frequency or width of the high speed clock.

A block diagram of a generalized memory or device interface is shown in Figure 47. The bottom half of the figure shows the accumulation of the serial input data from the SWI, and the top half of the figure shows the transmission of the serial output data to the SWI along with the return clock.

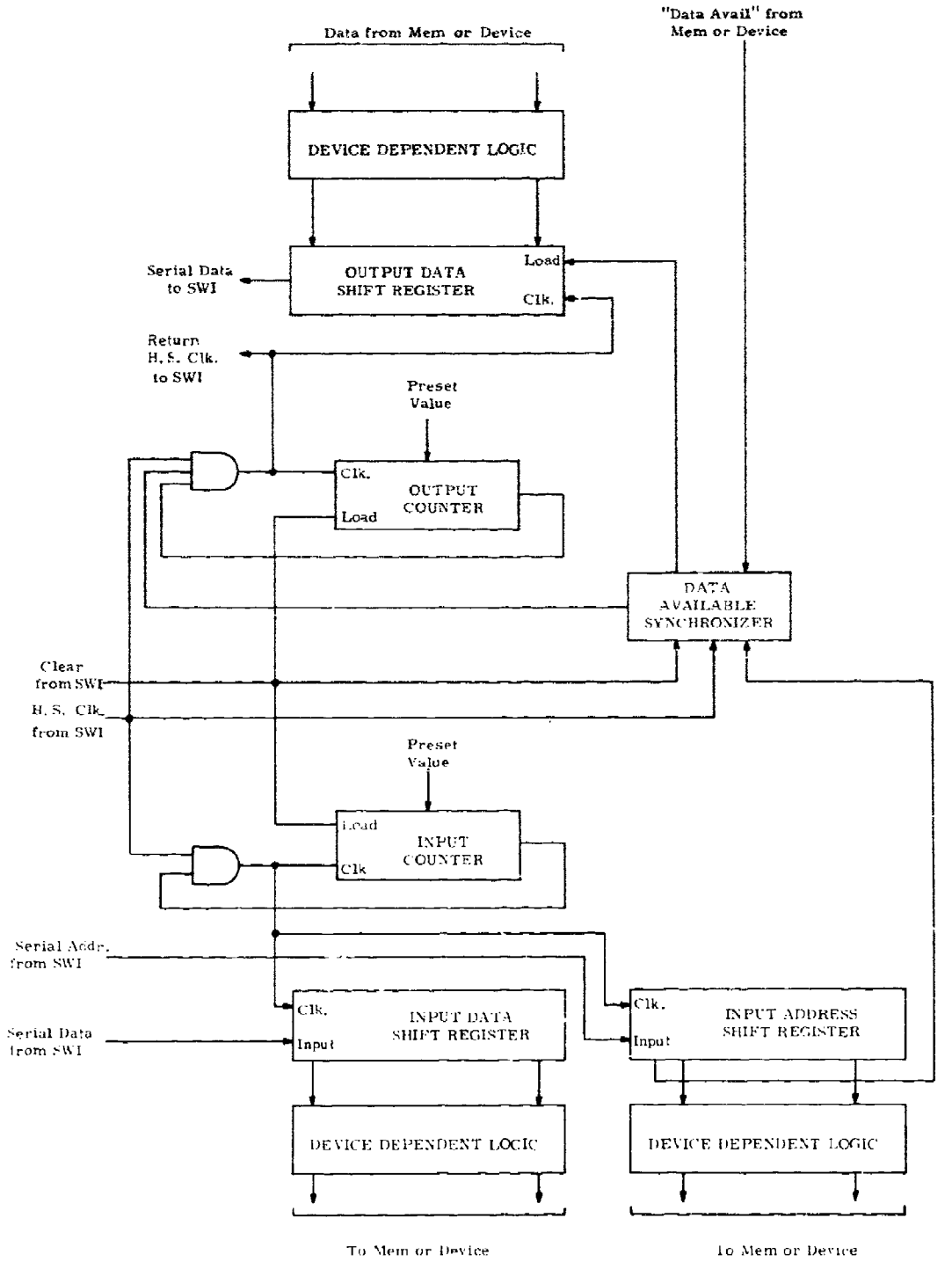


Figure 47. Memory Device Interface with SWI, Block Diagram

DEVICE INTERFACE OPERATION EXAMPLES

Line Printer

The printer is device No. 1 (i.e. the most significant three bits of the selected base register are 001). It is assumed that the appropriate locking to the printer will have been performed prior to initiating printer operations.

Line Printer Operation

The values of the bits of the MAR accompanying a DW or DR to the printer are interpreted as follows:

MAR 7 (LSB)		unused
MAR 6	=	0 for forms controls in six LSB's of MIR
	=	1 for character in six LSB's of MIR
MAR 5	=	0 when transferring characters
	=	1 when printing or using forms controls
MAR 0-4		unused

The following sequence will print a full 132 character line followed by a single space.

Printer/Interpreter Synchronization

To synchronize the Interpreter with the printer clock, a DR with controls bits 010 in the least significant three bits of the MAR is issued. This operation has no effect upon the printer, but causes the DDP to return an RDC on the trailing edge of the next printer clock.

Printer Buffer Loading

133 characters must be transferred into the print buffer. The last 132 of these will print from right to left on the line. The first character is totally ignored. Character transfer is initiated by a DW with control bits 010 in the least significant three bits of the MAR. The 6 least significant bits of the MIR which are present at the end of the Fetch Phase of the instruction containing the DW are transferred into the printer buffer as a BCL character. After the character has been accepted by the printer an RDC is returned. In the same clock in which this RDC is received, a DW containing the next character must be initiated as described below under "Timing Considerations". The first DW in the sequence of 133 should wait for the RDC which is received from the synchronizing DR.

Print Initiation

When the RDC from the 133rd character transfer is received, a DW with control bits 100 in the MAR and all zeros in the MIR is issued. This control will cause the printer to print the buffer.

Single Space Initiation

When the RDC from the print is received, a DW with control bits 100 in the MAR, a one in the least significant bit of the MIR and zeros in all other MIR bits is issued. This will cause a single space. Other spacing can be done instead by placing other values in the six least significant bits of the MIR. The format of the MIR for forms control is as follows.

MIR	31 (LSB)	PSSL	ONE for single space
	30	PDSL	ONE for double space
	29	FC1L	Format controls for variable spacing (110000 for bottom of form) (000100 for top of form)
	28	FC21	
	27	FC4L	
	26	FC8L	
	25	unused	
	24	unused	

Delay for Printing/Spacing

A delay of approximately 150 milliseconds must elapse prior to filling the buffer for the next line. With this delay a continuous printing speed of 400 lines per minute can be maintained.

Status Information

When RDC is returned from either a DW or DR, a BEX instruction will bring status information into the B register as follows:

B	31 (LSB)	PRRL	Ready, ZERO when ready
	30	PAML	Paper Motion, ZERO when paper in motion or print cycle in progress
	29	PCYL	Cycle, ZERO when print cycle in progress
	28	EOPL	End of Page, ZERO when end of page sensed
	27	PPEL	Parity Error, ZERO for transmission parity and/or print counter sync error
	26	PFCP	Final Character Pulse, ZERO after last character of line
	25	unused	
	24	unused	

If the program does not test for the not ready condition and the stop button is pushed, the program will continue to send and receive information from the DDP although no actual printing will occur and data will be lost. To control printing, the ready level need only be tested once each line prior to filling the print buffer, since the not ready condition (STOP light on) cannot occur after a load buffer instruction until the line has been printed.

Timing Considerations

Loading of the printer buffer involves the transfer of a BCL character from an Interpreter to the printer every 10 microseconds. Because the data transferred should be present on the printer input lines for at least 9 microseconds prior to its acceptance by the printer (for reliable settling), only 1 microsecond should elapse between the termination of transfer of one character and the initiation of transfer of the next. If less than 9 microseconds are allowed for settling, some bit positions with value 0 will be read incorrectly as 1, thus causing random incorrect characters to be printed.

The transfer of data from the printer input lines into the printer buffer occurs every 10 microseconds on the trailing edge of the printer clock pulse. This clock pulse also causes the status bit to be sent to the SWI from the printer DDP. After the last of these data bits has been received by the SWI, the return of an RDC to the Interpreter is initiated. Because of resynchronization delays in the SWI, this RDC will not be detected by the Interpreter until 2 1/2 clocks later on the average. The Interpreter must then issue a new DW containing the next character to be loaded. This character will begin transferring into the DDP at the end of the clock in which the DW is initiated. The transfer will take 4 high-speed clocks to complete, at which time the new character will be present on the printer input lines, and will begin settling. The entire process described here should occur within 1 microsecond in order that 9 microseconds will be available for settling.

Card Reader

The card reader is device No. 2 (i.e. the most significant three bits of the selected base register are 010.) To be used the card reader must be locked to an Interpreter and the base register must select the card reader. Upon successful completion of DL, an SAI is returned to the Interpreter.

To start up a card reader it must be sent proper bits in a DW or a DR instruction. The values of the MAR accompanying the DW are interpreted as follows:

Least significant bit:	0 Don't return data to SWI
	1 Return data to SWI

The LSB is normally a 1, the 0 value allows skipping cards or testing card reader mechanical functions without data or RDC returns to the SWI.

Next to LSB:	0 Return character bits as data
	1 Return status bits as data
Third from LSB:	0 Read as BCL
	1 Read as Hollerith

This Hollerith reading function is not wired on the present card reader DDP for the 6 high rows (11, 12, 0, 1, 2, 3); only the hole pattern for the 6 low rows (4, 5, 6, 7, 8, 9) are returned.

Fourth from LSB: 0 Don't operate card reader
1 Operate card reader

The 0 value allows checking of DDP functions without the noise of the card reader.

These control bits apply to the DW which they accompany and to all following DR's for this card reader until changed by another device write. Upon completion of a DW, data is returned to the Interpreter via the SWI and an RDC occurs to mark the end of the data reply for the write. When status is selected as data, the status returned with the DW (and subsequent DR's, if any) is valid, however the character returned with the DW is likely to be meaningless. The status bits returned are these:

LSB:	CRL:	Ready, ONE for ready
	CCL:	Present, ONE for duration of each card
	CREL:	Error, ONE for reader detected error
	CRCL:	Start, ONE for START button Not operated
	EOF:	End of File, ONE for Hopper Not Empty or for EOF button Not operated (ZERO for Empty Hopper and EOF Button operated.)
	Not used:	Zero
	Not used:	Zero
MSB:	Not used:	Zero

Immediately upon receipt of a DW containing bits set to operate it, the card reader begins to read cards at its maximum rate. Since the DDP for the card reader has but a 1 column buffer, it is necessary for the program in the Interpreter to send a DR instruction for each column. The synchronization of DR's and column reads in the DDP is as follows: Case 1. The DR arrives at the DDP before the column read is ready: The DR waits at the DDP until the column read is ready; then transmits data and return clocks to the Interpreter. If during this wait another SWI operation is invoked which returns as RDC before the column read is ready, the DR in the card reader DDP is lost and a new device read must be sent to the card reader to capture the data of this column. Upon sending the data of this column, the state of the DDP is set to show no column read ready. Case 2: The DR arrives at the DDP after the column read is ready. The DR immediately returns data and return clocks to the Interpreter and sets the state of the DDP to show no column read ready. If during the actions of this DR, another SWI operation is invoked which returns an RDC before the DR is complete, the DR in the card reader DDP is lost, the card column is lost and the control sequence of the DDP is confused.

SECTION VII

INTERPRETER MICROPROGRAMMING

Microprogramming is that procedure the designer uses to specify the action, function, and state of each of the Interpreter logic elements during every clock time. (A historical background of microprogramming is given in appendix I). In this sense, microprogramming replaces the function of hardware sequential logic used to cause the machine to execute an instruction requiring more than one clock time. Thus, microprogramming is essentially similar to sequential logic design. However, no logic (hardware) is added in the sequential logic design, but rather the existing registers, data paths, and control gates are used in a specific order to bring about the desired logical result.

The pattern of ones and zeros in the Microprogram Memory (MPM) and nanomemory (together with the data) determines the operation of the Interpreter. The microprogrammer is concerned with the generation of these patterns to provide the desired control functions. However, instead of actually writing these patterns, the microprogrammer is assisted by a microtranslator (or assembler) that allows him to write microinstructions mnemonically. The microtranslator then scans these instructions and produces the pattern of ones and zeros to be placed into the MPM and Nanomemories.

Figure 48 indicates how one can learn to microprogram the machine and the simplicity of the microprogram structure. The high degree of parallelism in the Interpreter is also evident from the powerful statements that can be expressed. For example, the following actions may be expressed and performed in one instruction:

test a condition (for either True or False)

set/reset a condition

initiate an external operation (e.g., memory read)

Type I – Use of nano memory (54 bits)

		A		B			
Nanobits	1-7 (7)	8-10 (3)	51-54 (4)	17-41 (25)	42-50 (9)	11-13 (3)	14-16 (3)
*; If	Condition	then Condition Adjust;	External	LU	MCU/CU;	True Succ	else False Succ
	GC1/2	Set LC1/2/3	Main Memory: Read/Write	A Select	Control for:	wait	wait
	LC1/2/3	Set GC1/2		B Select		step	step
	SAI	Set INT		Z Select	AMPCR	save	save
	EX1/2	Reset GC		Adder Function	BR1/2	skip	skip
	MST		Device: Read/Write Lock/Unlock	Shift Select	MAR	jump	jump
	LST			Destination(s)	CTR	exec	exec
	ABT				SAR	call	call
	AOV					retn	retn
	COV		Load MPM				
	INT		Load Nano				
	RDC						

*Groups A and B may be executed either conditionally as shown or unconditionally by being placed before condition test.

Type II – Loads any of 3 specified registers (no nano memory access, step successor)
Four variations

$k \rightarrow$ SAR
$k \rightarrow$ LIT
$k \rightarrow$ AMPCR
$k_1 \rightarrow$ SAR; $k_2 \rightarrow$ LIT

Figure 48. Microinstruction Types

perform an add operation

shift the result of the add

store the results in a number of registers

increment a counter

complement the shift amount

choose the successor microinstruction

It is also possible to perform these operations either conditionally or unconditionally as suggested in Figure 48. The group A and group B portions (either, neither, or both) of the microinstruction may be placed before the condition test portion of the instruction. This will result in that portion (A and/or B) being performed unconditionally.

The following four microinstruction examples illustrate both the parallelism and the conditional/unconditional properties of the microinstructions.

(1) If NOT LST then Set LC1, MR1; A1 + B + 1 C→A2, MIR, CSAR, INC; Step else jump

(2) Set LC1, MR1; If NOT LST then A1 + B + 1 C→A2, MIR, CSAR, INC; Step else Jump

(3) A1 + B + 1 C→A2, MIR, CSAR, INC; If NOT LST then Set LC1, MR1; Step else Jump

(4) Set LC1, MR1; A2 + B + 1 C→A2, MIR, CSAR, INC; If NOT LST then Step else Jump

In (1) the LST bit is tested and if not true, the local condition 1 (LC1) is set, memory read is initiated (MR1), the function A1 + B + 1 is performed in the adder, the adder output is shifted circular and the result stored in both the A2 and MIR registers, the content of the shift amount register is complemented (CSAR), the counter is incremented (INC), and the true successor (STEP) is selected. If the LST bit is true, none of these operations are performed and the false successor (JUMP) is executed.

In (2) the LC1 is set and the memory read is initiated (MR1) unconditionally (i.e., without considering the LST bit). The remaining functions are conditionally performed as in (1).

In (3), the functions A1 + B + 1 C→A2, MIR, CSAR, INC are performed unconditionally but set LC1 and MR1 are performed conditionally.

In (4) the functions Set LC1, MR1, A1 + B + 1 C→ A2, MIR, CSAR, INC are all performed unconditionally and only the successors Step and Jump depend upon the LST test.

TRANSLANG FOR MICROPROGRAMMING

The TRANSlator LANGuage (TRANSLANG) program is an assembler for Interpreter microprograms. The complete syntax of TRANSLANG is given in Appendix IV. It employs a vocabulary of reserved words and symbols used to develop a microprogram and its corresponding table of nanoinstructions. Reserved words and symbols are grouped as defined in this report to form microinstructions and programs. The reserved words are summarized in Appendix V.

Two versions of TRANSLANG exist for the aerospace multiprocessor. One version is written in Burroughs Compatible ALGOL which can run on both Burroughs B 5500 and B 6700 systems. This TRANSLANG is described in this section and in more detail in Burroughs Microprogramming Manual for Interpreter Based Systems, TR70-8. The second version is written in FORTRAN for the CDC 6600, and is described in A FORTRAN Microprogram Translator, an Air Force Institute of Technology thesis GGC/EE/72-2. The TRANSLANG syntax and semantics for the FORTRAN version are the same as that described here and in TR70-8 with the exceptions listed in an appendix to the thesis.

Each TRANSLANG line corresponds to one microinstruction which is the set of Interpreter functions performed in parallel at each machine clock. The constructs include iterative mechanisms, I/O, Boolean, logical and computational operations, control transfers and assignment functions. In order to provide control points for transfer operations, each instruction may be labeled with a symbolic microaddress.

The INSERT function has been included to allow for the use of a macro library of previously debugged microprograms.

Conventions in Language Description

Backus-Naur form (BNF) is used as the metalanguage to define the syntax of TRANSLANG. The following BNF symbols are used:

1. $\langle\ \rangle$ Left and right broken brackets are used to bracket the names of syntactic categories.
2. ::= Colon colon equal means "is defined as" and separates the name of the syntactic category from its definition.
3. | Bar separates alternative definitions of a syntactic category.
4. { } Left and right braces enclose an English language description of a syntactic unit.

Any character or symbol in a metalanguage formula which is not a metalanguage symbol and is not enclosed within matching braces or broken brackets, denotes itself.

Basic Elements

⟨Letter⟩ ::=	A | B | C | D | E | F | G | H | I | J | K | L | M | N | O P | Q | R | S | T | U | V | W | X | Y | Z
⟨Digit⟩ ::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨Hex Digit⟩ ::=	⟨Digit⟩ | A | B | C | D | E | F
⟨Symbol⟩ ::=	, | ; | + | - | : | = | % | (|) | *
⟨Single Space⟩ ::=	{ One horizontal blank position }
⟨Space⟩ ::=	⟨Single Space⟩ | ⟨Space⟩ ⟨Single Space⟩
⟨Assignment Op⟩ ::=	=: | =
⟨Character⟩ ::=	⟨Letter⟩ | ⟨Digit⟩ | ⟨Single Space⟩ | ⟨Symbol⟩
⟨Comment Character⟩ ::=	⟨Character⟩ | . | # | & | $ | [|] | \ | /
⟨Empty⟩ ::=	{ The null string of characters }

Semantics

TRANSLANG uses a character set of 56 characters including ⟨single space⟩, 8 of which are only used in comments. All letters are upper case.

Spaces - No space may appear between the letters of a reserved word or within an ⟨Assignment Op⟩ ; otherwise, they will be interpreted as two or more elements. Spaces are used as a delimiter to separate reserved words, labels, or integers. Spaces may appear between any two basic components without affecting their meaning, where basic components indicate reserved words, symbols, or labels.

Parentheses - The parentheses are treated as spaces. They are used for the convenience of the microprogrammer to make code more readable. (E.g. instruction elements which are irrelevant to the current instruction but are used only to allow shared use of a nanoinstruction by several microinstructions.)
Parentheses do not imply precedence.

LITERAL ASSIGNMENT INSTRUCTION

⟨Literal Assignment⟩ ::= ⟨Literal⟩ ⟨Assignment Op⟩ AMPCR |
⟨Literal⟩ ⟨Assignment Op⟩ SAR |
⟨Literal⟩ ⟨Assignment Op⟩ SAR;
⟨Literal⟩ ⟨Assignment Op⟩ LIT |
⟨Literal⟩ ⟨Assignment Op⟩ LIT;
⟨Literal⟩ ⟨Assignment Op⟩ SAR |
⟨Literal⟩ ⟨Assignment Op⟩ LIT
⟨Literal⟩ ::= ⟨Integer⟩ | COMP ⟨Integer⟩ | ⟨Label⟩ | ⟨Label⟩ -1
⟨Integer⟩ ::= ⟨Digit⟩ | ⟨Digit⟩ ⟨Integer⟩
⟨Label⟩ ::= ⟨Letter⟩ | ⟨Label⟩ ⟨Letter⟩ | ⟨Label⟩ ⟨Digit⟩

Semantics

A ⟨Literal Assignment⟩ becomes a type II microinstruction for an Interpreter. This microinstruction contains the literal value(s) and specifies the receiving register(s).

		Width, bits
AMPCR	Alternate Micro Program Count Register	12
SAR	Shift Amount Register	5
LIT	Literal Register	8

The registers may be individually loaded or both the SAR and the LIT may be loaded in the same microinstruction.

An ⟨Integer⟩ is non-negative and in the range of the intended receiving register(s). COMP ⟨Integer⟩, if the receiving register is LIT or AMPCR, takes the one's complement of the ⟨Integer⟩, then takes the number of bits indicated by the width of the receiving register. COMP ⟨Integer⟩, for SAR, creates the appropriate word length complement. (This is two's complement for the 32-bit wide LSI Interpreter). The encoded value is used in the SAR field. The sucessor of a ⟨Literal Assignment⟩ is implicitly STEP.

Labels used in a program may be chosen freely except for the reserved words of TRANSLANG. The reserved words are given in Appendix V. A label must start with a letter which can be followed by any combination of letters or digits. No spaces or symbols may appear in a label. A label can be as little as one letter and as long as 15 letters and digits. The same label may not be used to locate more than one instruction in the same program. See the INSERT function subsequently described for allowable nesting of labels when subprograms are inserted. The normal use of a label with a ⟨Literal Assignment⟩ is as ⟨Label⟩ -1 since control transfers occur to the indicated location +1 (or +2 if a return is used).

Examples

```
5=: SAR                      % converted for proper logic unit width
COMP 8 =: SAR; 13=: LIT      % in one microinstruction
COMP.0 =: LIT                % same as 255=:LIT
START =: AMPCR               % JUMP to START +1; RETN to START + 2
LOOP-1=: AMPCR               % JUMP to LOOP; RETN to LOOP + 1
```

N INSTRUCTION

⟨N Instruction⟩ ::= ⟨Unconditional Part⟩ ⟨Conditional Part⟩

⟨Unconditional Part⟩ ::= ⟨Component List⟩

⟨Component List⟩ ::= ⟨Component⟩ | ⟨Component List⟩ ; ⟨Component⟩ | ⟨Empty⟩

⟨Component⟩ ::= ⟨Ext Op⟩ | ⟨Logic Op⟩ | ⟨Successor⟩

⟨Conditional Part⟩ ::= ⟨If Clause⟩ ⟨Cond Comp List⟩ ⟨Else Clause⟩ | ⟨If Clause⟩ | ⟨When Clause⟩ ⟨Cond Comp List⟩ | ⟨Empty⟩

⟨Cond Comp List⟩ ::= THEN ⟨Component List⟩

Semantics

An ⟨N Instruction⟩ becomes a Type I microinstruction containing an address of a nano instruction. If an identical nano instruction already exists, the microaddress will point to the single copy of the nano instruction. If the nano instruction is new, the address will be to the next unused nano address. The operations indicated in the <N Instruction> are entered into this nano location.

Restrictions

1. At most one ⟨Ext Op⟩ - either unconditional or conditional.
2. At most one ⟨Logic Op⟩ - either unconditional or conditional.
3. At most either one unconditional successor, or one conditional successor in the ⟨Cond Comp List⟩ and one in an ⟨Else Clause⟩.

The ⟨Unconditional Part⟩ is always executed. In the ⟨Conditional Part⟩ if the condition resulting from the ⟨If Clause⟩ or ⟨When Clause⟩ is true then the components in the ⟨Cond Comp List⟩ are executed, otherwise only the ⟨Else Clause⟩ is executed.

Examples (to be subsequently explained)

Unconditional Part, Component List:

SET GC1

MR2

RESET GC, DR2

A2 AND B001 =: A1

A1 + B IC R =: A2, BEX, LMAR

JUMP

DL1; 0=: A2; SKIP

Conditional Part:

IF AOV THEN A1 + 1 =: A1 ELSE SKIP

IF NOT ABT THEN SET LC2; SKIP ELSE SAVE

WHEN RDC THEN MR2; BEX, INC

N Instruction:

WHEN RDC THEN BEX

SET LC1; IF SAI THEN B ADL LIT = A3, BBE

CONDITION

⟨If Clause⟩ ::=	IF ⟨Condition⟩
⟨Condition⟩ ::=	⟨Not⟩ ⟨Cond⟩
⟨Not⟩ ::=	NOT | ⟨Empty⟩
⟨Cond⟩ ::=	LST | MST | AOV | ABT | COV | SAI | RDC | EX1 | EX2 | HOV | ⟨Cond Adjust Bit⟩
⟨When Clause⟩ ::=	WHEN ⟨Condition⟩
⟨Else Clause⟩ ::=	ELSE ⟨Sucessor⟩ | ⟨Empty⟩
⟨Cond Adjust Bit⟩ ::=	INT | LC1 | LC2 | LC3 | GC1 | GC2

Semantics

Each ⟨N Instruction⟩ performs a test on the Boolean value of one ⟨Cond⟩ or its complement. The Boolean value of the result is ⟨Condition⟩. If this value is true, the ⟨Cond Comp List⟩ is executed and the sucessor from this list is used to determine the next microinstruction. Otherwise the successor in the ⟨Else Clause⟩ is used to determine the next microinstruction address. See the subsequent discussion of successor.

A ⟨When Clause⟩ is a synonym for an ⟨If Clause⟩ with the same ⟨Condition⟩ and an ⟨Else Clause⟩ of ELSE WAIT. An empty ⟨Else Clause⟩ is equivalent to ELSE STEP.

In the absence of an ⟨If Clause⟩ or ⟨When Clause⟩, an implied ⟨If Clause⟩ of IF NOT GC1 is inserted. This changes no condition bit. If does cause unconditional initiation of a ⟨Logic Op⟩ and hence completion of the prior ⟨Logic Op⟩.

With the exception of the two global condition bits, testing a condition bit causes the bit to be reset. However, all condition bits are set dominant. Therefore in case a condition bit is being tested at the same time it is being set, the condition bit will not be reset. The least and most significant bits out of the adder, the adder overflow, and the adder bit transmit are levels and not condition bits. The conditions that may be tested (Table III) are the following:

SAI — Switch Interlock Accepts Information

Following memory or device operation, indicates that connection to the addressed memory or device is completed through the switch interlock and that the MAR and MIR may be changed.

RDC — Read Complete, or Requested Device Completes

Following memory read or device read, indicates that data will be available for entry to B in the next clock. Following device write, indicates completion of write.

COV — Counter Overflow

Following or concurrent with increment counter INC, indicates counter is overflowing or has already overflowed from all ones (255) to all zeros.

LC1 — Local Condition 1

Tests and resets local Boolean condition bit LC1.

LC2, LC3 — Local Conditions 2 and 3
Same as LC1

Table III. Set and Reset of Conditions

BIT	SET	RESET
AOV	Dynamic Adder State - (Overflow)	#
ABT	Dynamic Adder State - (Adder bit transmit)	#
LST	Dynamic Adder State - (Least Significant Bit of Adder Output)	#
MST	Dynamic Adder State - (Most Significant Bit of Adder Output)	#
COV	Overflow when Counter is Incremented	Reset by loading counter or by testing
GC1	SET GC1 providing no other Interpreter has GC1 set, or no higher priority Interpreter is concurrently doing SET GC1	RESET GC
GC2	SET GC2 similar to GC1	RESET GC
INT	Set INT executed in any Interpreter	Reset by testing
LC1	SET LC1	Reset by testing
LC2	SET LC2	Reset by testing
LC3	SET LC3	Reset by testing
RDC	By memory at completion of memory or device read	Reset by testing
SAI	By switch interlock when data received from MAR and MBR	Reset by testing
EX1	By requests from devices	Reset by testing
EX2	By requests from devices	Reset by testing
HOV	Horn overflow	Reset by testing

#Recomputed each clock time

GC1 } Global Conditions 1 and 2
GC2 } Tests but does not reset global condition bit GC1. See the description of the set and reset operation for further explanation of global condition bits.

INT Inter-Interpreter Interrupt

Tests and resets the local copy of the inter-Interpreter interrupt.

EX1 External Conditions 1 and 2
EX2 Test and reset interrupts (usually the OR of interrupts from several devices) from external devices (local copy). These are presently wired to switches in the aerospace multiprocessor.

HOV Horn Overflow

Indicates that no ⟨ Ext Op ⟩ has occurred during a period of 2^{20} Interpreter clocks, (approximately 1 second for a 1 MHz Interpreter clock). This is used for detection of a failed memory module or devices and will force a STEP in the microprogram at the same time this condition bit is set.

The following four logic unit conditions are dynamic and indicate the result output from the adder using the execution phase commands from the previous instruction which had logic unit operation, and using the current values of the adder inputs. These conditions are sustained until execution of another instruction involving the logic unit, and may be tested by that instruction. A type II instruction loading the LIT or AMPCR may change the value of an adder input selected in the ⟨Z Select⟩ and hence change the value of any of these conditions.

AOV Adder Overflow

State of the carry out of the most significant bit of the adder.

LST Least Significant

State of the least significant bit of the adder output.

MST Most significant

State of the most significant bit of the adder output.

ABT Adder bit transmit

This condition is true (one) if and only if the adder output is all ones or all zeros depending on the specific operator performed. (See Appendix III).

Examples

IF NOT LC1

WHEN SAI

ELSE CALL

EXTERNAL OPERATIONS

⟨Ext Op⟩ ::= ⟨Mem Dev Op⟩ | ⟨Set Op⟩ |
⟨Mem Dev Op⟩ , ⟨Set Op⟩ |
⟨Set Op⟩ , ⟨Mem Dev Op⟩ | ⟨Empty⟩

⟨Mem Dev Op⟩ ::= MR1 | MR2 | MW1 | MW2 | DL1 | DL2 | DR1 | DR2 |
DW1 | DW2 | DU1 | DUZ | LDM | LDN

⟨Set Op⟩ ::= SET ⟨Cond Adjust Bit⟩ | RESET GC

Semantics

The external operations are ⟨N Instruction⟩ functions which, if explicity present, affect the operations external to the Interpreter logic. An ⟨Ext Op⟩ may be specified as either conditional or unconditional as it appears in at most one of the ⟨Unconditional Part⟩ or ⟨Conditional Part⟩.

The memory or device operations ⟨Mem Dev Op⟩ are used to transfer data between the Interpreter and S memory or a peripheral device. Address source registers for those operations are the combination of either BR1 or BR2 with MAR, indicated respectively by MAR1 or MAR2. The MAR holds the less significant part of the address. The memory or device operations are described in detail in Section VI. The explicit memory or device operations follow.

MR1 Memory Read 1
Read data from S memory address specified in MAR1

MR2 Memory Read 2
Read data from S memory address specified in MAR2

MW1 Memory Write 1
Write data from MIR to S memory address specified in MAR1

MW2 Memory Write 2
Write data from MIR to S memory address specified in MAR2

LDM — Load a microinstruction from the least significant 16 bits of the MIR into a word in microprogram memory (MPM) as specified by AMPCR.

LDN — Load least significant 16 bits of MIR into the nanoword as specified by the nanoaddress contained in the microprogram word being specified by AMPCR. The syllable of the nanoword loaded is specified by the two bits next to the least significant bit in the MAR.

DL1 — Device Lock 1 Request
Reserve the device or memory module named in MAR1 for use by this Interpreter.

DL2 — Device Lock 2 Request
Reserve the device or memory module named in MAR2 for use by this Interpreter.

DR1 — Device Read 1
Read data from device named in MAR1

DR2 — Device Read 2
Read data from device named in MAR2

DW1 — Device Write 1
Write data from MIR to the device named in MAR1

DW2 — Device Write 2
Write data from MIR to the device named in MAR2

DU1 — Device Unlock 1
Release the locked device named in MAR1

DU2 — Device Unlock 2
Release the locked device named in MAR2

The set and reset operations are used to set and reset condition bits. The inter-Interpreter interrupt INT, is used for communication among (to signal) all Interpreters of the aerospace multiprocessor. The global conditions, GC1 and GC2, are used as Boolean semaphores to guarantee mutual exclusion for critical sections of microprograms and to prevent simultaneous access to shared data. The local condition bits are Boolean variables local to each Interpreter. The INT and local condition bits are reset (within the local Interpreter only) by testing. The explicit test and reset operations follow. If no ⟨Set Op⟩ is present, none is done.

SET INT — Interrupt Interpreters

Causes the interrupt bit to be set in all Interpreters. Each Interpreter resets its own bit by testing it. Setting occurs after testing should both occur in the same nano-instruction.

SET LC1 — Set the first local condition bit

Causes the setting of the LC1 bit in the condition register. Setting occurs after testing should both occur in the same nano-instruction. Both set and test of LC1 occur during the fetch phase of a microinstruction.

SET LC2 — Set the second local condition bit

Same as for LC1 replacing LC1 by LC2.

SET LC3 — Set third local condition bit

Same as for LC1 replacing LC1 by LC3.

SET GC1 — Set first global condition bit request

Requests that the GC1 bit in the requesting Interpreter be set if a GC1 bit is not already set in another Interpreter or is not requesting to be set simultaneously by a higher priority Interpreter. For all Interpreters in a multiprocessing system at most one will have GC1 set. GC1 is set at the end of the phase after the fetch phase if no conflict occurs. A request lasts for one clock.

SET GC2 — Set second global condition bit request

Same as for GC1 replacing GC1 by GC2.

RESET GC — Resets the global condition bits

Causes GC1 and GC2 to be reset in the issuing Interpreter.

Examples

```
MR2

SET LC1

DR2, RESET GC
```

LOGICAL OPERATIONS

⟨Logic Op⟩ ::= ⟨Adder Op⟩ ⟨Inhibit Carry⟩⟨Shift Op⟩⟨Destination List⟩

⟨Adder Op⟩ ::= 0 | 1 | ⟨Monadic⟩ | ⟨Dyadic⟩ | ⟨Triadic⟩ | ⟨Empty⟩

⟨Monadic⟩ ::= ⟨Not⟩ ⟨A Select⟩ | ⟨Not⟩ ⟨B Select⟩ |
⟨Not⟩ ⟨Z Select⟩

⟨Not⟩ ::= NOT | ⟨Empty⟩

⟨Dyadic⟩ ::= ⟨A Select⟩ ⟨Binary Op⟩ ⟨B Select⟩ |
⟨B Select⟩ ⟨Binary Op⟩ ⟨Z Select⟩ |
⟨A Select⟩ ⟨AZ Op⟩ ⟨Z Select⟩

⟨Binary Op⟩ ::= ⟨AZ Op⟩ | OR | NIM | IMP | NOR

⟨AZ Op⟩ ::= AND | XOR | EQV | NRI | RIM | NAN | ADD | + | ADL | CAD

⟨Triadic⟩ ::= ⟨Try Op⟩ ⟨A Select⟩ , ⟨B Select⟩ , ⟨Z Select⟩

⟨Try Op⟩ ::= TRY1 | TRY2 | TRY3 | TRY4 | TRY5

⟨Shift Op⟩ ::= R | L | C | ⟨Empty⟩

⟨Inhibit Carry⟩ ::= IC | ⟨Empty⟩

Semantics

The logical operations include those operations which occur within and affect the logic unit of the Interpreter. This group of operations may be specified as unconditional if placed before the ⟨If Clause⟩ of a conditional instruction and conditional if placed after the ⟨If Clause⟩.

The logic operations include the selection of adder inputs, the adder operation, the barrel switch operation, the destination specifications for the adder and BSW outputs and the controls for the literal, counter, and SAR registers.

Each instruction except the ⟨Literal Assignment⟩ contains an adder operation. If this is missing, the adder operation is assumed to be A + B (where A and B are zero). These adder operations may use input from one, two, or three different registers as specified in the ⟨A Select⟩ ⟨B Select⟩ ⟨Z Select⟩ parts of the instruction.

Monadic operators are those operators requiring one register input to the adder. The value of the selected register or the complement of the value may become the adder input depending on the ⟨Not⟩ function.

The dyadic operators are those adder operators that may occur between two registers. These include arithmetic as well as logical operators. The arithmetic operators may occur with sources selected from any two of the three inputs - A, B, and Z.

ADD | +	Add the two inputs to the adder.
ADL	Add the two inputs to the adder + 1
CAD	Add the two inputs to the adder in groups of 8 bits. Inhibit carries between 8 bit bytes.

All logical operators except four may occur between selections from any two registers (A + B, B + Z, or A + Z). The four exceptions that may not occur between an A and Z select are OR, NIM, IMP and NOR.

OR	Or	X OR Y produces $X \vee Y$
NIM	Not Imply	X NIM Y produces $X\overline{Y}$
IMP	Imply	X IMP Y produces $\overline{X} \vee Y$
NOR	Nor	X NOR Y produces $\overline{X \vee Y}$

All other logical operations may occur between any two of the three registers selected.

AND	And	X AND Y produces XY
XOR	Exclusive Or	X XOR Y produces $\overline{X}Y \vee X\overline{Y}$
EQV	Equivalence	X EQV Y produces $XY \vee \overline{X}\overline{Y}$
NRI	Not Reverse Imply	X NRI Y produces $\overline{X}\ Y$
RIM	Reverse Imply	X RIM Y produces $X \vee \overline{Y}$
NAN	Not And	X NAN Y produces $\overline{X}\ \overline{Y}$ or $\overline{X} \vee \overline{Y}$

$\overline{X}$ means (ones) complement of X
precedence is complement done before AND done before OR

The triadic operators are those operators requiring three inputs to the adder (i.e., A, B, and Z). These are available in the Interpreter and may be used with the following notation:

TRY1 A, B, Z produces $\overline{A}\ \overline{B}\ Z \vee A\ \overline{B}\ \overline{Z}$

TRY2 A, B, Z produces $\overline{A}\ Z \vee B\ \overline{Z}$

TRY3 A, B, Z produces $A \vee B \vee \overline{Z}$

TRY4 A, B, Z produces $AZ \vee \overline{B}\ \overline{Z}$

TRY5 A, B, Z produces $AZ \vee BZ \vee \overline{A}\ \overline{B}\ \overline{Z}$

There are three shift operations, one of which may be selected each time an adder operator is used. These operations are R, L, or C.

R Right end off shift by amount in SAR

L Left end off shift by the two's complement of amount in SAR

C Circular shift right end around by amount in SAR

The carry bits may be inhibited, for all operations, between 8-bit bytes. IC inhibits carries.

Examples

```
NOT LIT =: A2

A1 ADL B R =: B

A2 + LIT =: SAR

DEC CTR

TRY1 A2, B110, CTR

   0 =:A3

   1 =:CTR

A2 + CTR IC R = A2,  BEX,  CTR,  CSAR
```

INPUT SELECTS

⟨A Select⟩ ::= A1 | A2 | A3 | 0 | ⟨Empty⟩

⟨B Select⟩ ::= B | B ⟨M⟩ ⟨C⟩ ⟨L⟩ | ⟨Empty⟩

⟨M⟩ ::= ⟨Gating⟩

⟨C⟩ ::= ⟨Gating⟩

⟨L⟩ ::= ⟨Gating⟩

⟨Gating⟩ ::= 0 | 1 | T | F

⟨Z Select⟩ ::= CTR | LIT | AMPCR | 0 | ⟨Empty⟩

Semantics

There are three A registers which may be used for data storage within an Interpreter. Any one of the A registers may be selected as input to the adder in an instruction. The B register is the primary interface for external inputs from main memory or devices. It also serves as input to the adder. The B register can be partitioned into three parts when it is selected as input to the adder. The partitions are as follows:

M	Most significant bit of B (left most bit)
C	Central bits of B (all but the end bits)
L	Least significant bit of B (right most bit)

When selecting the B register as input to the adder, each of the three parts may be independently specified as being either 0, 1, T, or F. A zero gating will cause that part to be all zeros. A one gating will cause that part to be all ones. A T gating will produce the true value of B for that part. An F gating will produce the complement value of B for that part. The B register and its gating is specified without embedded spaces. If no gating is specified when selecting B, then it is assumed that the true value of B is desired (i.e., BTTT).

There are three registers which make up the ⟨Z Select⟩ input to the adder. These are the counter (CTR), the literal (LIT) and the AMPCR. The counter register when used as input to the adder, is left justified with zero fill. The literal register, when used as input to the adder is right justified with zero fill. The AMPCR comes into the least significant 12 bits of the center 16 bits of the adder. The most significant 4 bits of the center 16 bits of the adder contain the binary value of the Interpreter number right justified in the 4-bit field. The rest of the adder is zero filled.

Examples

A1 + B + 1 IC R

A2 XOR CTR

BOTT AND LIT

DESTINATION OPERATORS

⟨Destination List⟩	::=	⟨Asgn⟩ ⟨Dest⟩ | ⟨Destination List⟩ ⟨Asgn⟩ ⟨Dest⟩ | ⟨Asgn⟩
⟨Asgn⟩	::=	, |=: | =
⟨Dest⟩	::=	A1 | A2 | A3 | MIR | BR1 | BR2 | AMPCR | ⟨Input B⟩ | ⟨Input Ctr⟩ | ⟨Input Mar⟩ | ⟨Input Sar⟩
⟨Input B⟩	::=	B | BEX | BAD | BC4 | BC8 | BMI | BBE | BBA | BBI | BAI | BBAI | B4I | B8I
⟨Input Ctr⟩	::=	CTR | LCTR | INC
⟨Input Mar⟩	::=	MAR | MAR1 | MAR2 | LMAR
⟨Input Sar⟩	::=	SAR | CSAR

Semantics

The destination operators explicitly specify registers in which changes are to occur at the end of a logic unit operation.

Restrictions:

1. At most one choice from each of ⟨Input B⟩, ⟨Input Ctr⟩, ⟨Input Mar⟩ and ⟨Input Sar⟩ is permitted.

2. If ⟨Input Ctr⟩ is LCTR then ⟨Input Mar⟩ may not be MAR, MAR1 or MAR2.

3. If ⟨Input Mar⟩ is LMAR then ⟨Input Ctr⟩ may not be CTR.

The principal data source is the barrel switch output. It is the only source for loading A1, A2, A3, MIR, BR1 and BR2. It provides one source for loading B,

CTR, MAR, SAR and AMPCR. These reserved words are also the register names. The bits used in these transfers are indicated below:

Destination Register	Barrel Switch Output Source Bits
A1	All
A2	All
A3	All
B	All
MIR	All
BR1	2nd least significant byte
BR2	2nd least significant byte
MAR	least significant byte
CTR	least significant byte (ones complement)
SAR	least significant 5 bits
AMPCR	least significant 12 bits

The B, MAR, CTR, SAR and AMPCR registers may have other inputs as well.

B Register – (B)

B	The barrel switch output is placed into B.
BEX	Data from the external source is placed into B.
BAD	The adder output is placed in the B register (short path to B).
BMI	The MIR content is placed in the B register independent of any concurrent change to the MIR.*
BC4	The duplicated complement of the 4-bit carries with zero fill is placed in the B register.**
BC8	The duplicated complement of the 8-bit carries with zero fill is placed in the B register.**
BBE	The barrel switch output ORed with the data from the external source is placed in the B register.

*When the MIR is one of the inputs to the B register, the input shift register from the Switch Interlock into the external input to B will be cleared to all zeros.

**Form of BC4, B4I, BC8, and B8I adder outputs for each 8-bit group:
The carries out of bits 2, 3, 4, 6, 7 and 8 are irrelevant.

Bit	1	2	3	4	5	6	7	8
Carries Out	u	-	-	-	-	v	-	-
B4I, BC4	o	o	$\bar{u}$	$\bar{u}$	o	o	$\bar{v}$	$\bar{v}$
B8I, BC8	o	o	$\bar{u}$	$\bar{u}$	o	o	o	o

BBA	The barrel switch output ORed with the adder output is placed in the B register.
BBI	The barrel switch output ORed with the MIR content is placed in the B register independent of any concurrent change to the MIR.*
BAI	The adder ORed with the MIR is placed in the B register.*
BBAI	The BSW ORed with the adder ORed with MIR is placed in the B register.*
B4I	The duplicated complement of the 4-bit carry ORed with MIR content is placed in the B register.*
B8I	The duplicated complement of the 8-bit carries with zero fill ORed with MIR content is placed in the B register.*

Memory Address Register - (MAR)

LMAR	The literal register content is placed in MAR.

Counter - (CTR)

LCTR	The one's complement of the literal register content is placed in CTR.
INC	Increment Counter by 1.

Shift Amount Register - (SAR)

CSAR	Complement (two's complement) prior content of SAR.

The Alternate Micro Program Count Register ⟨ AMPCR ⟩ may, during the same clock, receive input from the MPCR if the microprogram address control register content was CALL or SAVE. The MPCR source takes precedence over the AMPCR specification as a ⟨ Dest ⟩ .

Examples

```
=: B
=: CTR
=: A1, BEX, = MIR, LCTR, CSAR        % mixed use of, =, and =:
```

*When the MIR is one of the inputs to the B register, the input shift register from the Switch Interlock into the external input to B will be cleared to all zeros.

SUCCESSOR

⟨Successor⟩ ::= WAIT | STEP | SKIP | SAVE | CALL | EXEC | JUMP | RETN

Semantics

Each ⟨N instruction⟩ specifies 2 successors explicitly or implicitly, indicating the control to be used for the next microinstruction selection. A ⟨Successor⟩ in the ⟨Unconditional Part⟩ results in the 2 successors being identical. Otherwise one or two successors may appear in the ⟨Conditional Part⟩. The eight choices for each successor are described below and in Table IV.

WAIT	Repeat the instruction in the microprogram count register (MPCR).
STEP	Step to the next instruction in sequence from MPCR.
SKIP	Skip to the second next instruction in sequence from MPCR.
SAVE	Step and save current MPCR address in AMPCR.
CALL	Transfer control to AMPCR + 1 address, save current MPCR in AMPCR.
EXEC	Execute instruction in AMPCR + 1, proceed as specified in the executed instruction.
JUMP	Transfer control to AMPCR + 1 address.
RETN	Transfer control to AMPCR + 2 address.

Any successor not explicitly stated is STEP by default. All successors except EXEC place the resulting microprogram address in MPCR.

Each ⟨Literal Assignment⟩ instruction has an implicit successor of STEP.

The AMPCR normally contains the address of an alternative instruction (usually label-1). The AMPCR load of the current content of the MPCR from a CALL or SAVE takes precedence over a ⟨Literal Assignment⟩ into AMPCR in the dynamically next microinstruction. It also takes precedence over an explicit ⟨Dest⟩ of AMPCR from the ⟨Logic Op⟩ in progress.

Table IV. Microprogram Memory Addressing

Successor Command	Successor M-instruction Address	Next Content of MPCR will be	Next Content of AMPCR will be
WAIT	MPCR	MPCR	*
STEP	MPCR+1	MPCR+1	*
SKIP	MPCR+2	MPCR+2	*
SAVE	MPCR+1	MPCR+1	MPCR
CALL	AMPCR+1	AMPCR+1	MPCR
EXEC	AMPCR+1	MPCR	*
JUMP	AMPCR+1	AMPCR+1	*
RETN	AMPCR+2	AMPCR+2	*

*Not changed by successor specification

Examples

WAIT

JUMP

PROGRAM STRUCTURE

⟨Program⟩ ::= ⟨Program Name Line⟩ ⟨Body⟩ ⟨End Line⟩

⟨Program Name Line⟩::= PROGRAM ⟨Program Name⟩ ⟨Start Address⟩

⟨Program Name⟩ ::= ⟨Label⟩

⟨Start Address⟩ ::= ADR ⟨Hex Address⟩ | ⟨Empty⟩

⟨Hex Address⟩ ::= ⟨Hex Number⟩

⟨Hex Number⟩ ::= ⟨Hex Digit⟩ | ⟨Hex Number⟩ ⟨Hex Digit⟩

⟨Body⟩ ::= ⟨Statement⟩ | ⟨Comment⟩ | ⟨Body⟩ ⟨Statement⟩ | ⟨Body⟩ ⟨Comment⟩

⟨Statement⟩ ::= ⟨Label Part⟩ ⟨Line⟩ ⟨% Comment⟩

⟨Comment⟩ ::= COMMENT ⟨Comment Words⟩ ;

⟨Label Part⟩ ::= ⟨Label⟩ : | ⟨Empty⟩

⟨Line⟩ ::= ⟨Label Constant⟩ | ⟨Start Address⟩ | ⟨Insert⟩ | ⟨Instruction⟩

⟨Label Constant⟩ ::= ⟨Label⟩ * ⟨Integer⟩

⟨Insert⟩ ::= INSERT ⟨Label⟩ ⟨Start Address⟩

⟨% Comment⟩ ::= % ⟨Comment Words⟩ | ⟨Empty⟩
⟨Comment Words⟩ ::= ⟨Comment Character⟩ |
⟨Comment Words⟩ ⟨Comment Character⟩

⟨Instruction⟩ ::= ⟨Label Part⟩ ⟨Literal Assignment⟩ |
⟨Label Part⟩ ⟨N Instruction⟩

⟨End Line⟩ ::= END

Semantics

A file containing a source program must have a ⟨Label⟩ or 6 or less alphanumeric characters. Each record on this file contains 72 data characters (plus eight for sequence numbers, which is optional for the microtranslator). One ⟨ Statement ⟩ of source program is written per record.

The first record is the ⟨Program Name Line⟩. It contains the program internal name and possible a starting address for a microprogram. The program internal name should be the same as the file name. Only the file name has any external significance. An empty ⟨Start Address⟩ means start with zero for the first microinstruction of the program. A non-empty start address becomes a hexidecimal absolute microprogram address. The body of a program contains one or more statements. Following the body is the ⟨End Line⟩ containing END. Each successive statement containing an ⟨Instruction⟩ normally becomes the next microaddress. Addresses strictly increase through a program. If a ⟨Start Address⟩ is greater than the next address in the program sequence, microinstructions composed of all zeros are used to fill in the locations between the addresses in the output file. A ⟨Start Address⟩ less than the next address in the program sequence causes an error.

A label is defined for use in two ways. A ⟨Label Constant⟩ permits a ⟨Label⟩ to be declared to be an ⟨Integer⟩. Subsequent use of that label is replaced by the Integer. Use of a ⟨Label Constant⟩ prior to declaration is an error. A label is also defined upon occurrence in a ⟨Label Part⟩ in which case it serves as a symbolic reference to a particular line.

An ⟨Insert⟩ is used to allow a user access to his files outside the program file. When the ⟨Insert⟩ is recognized, the microtranslator extracts from the users files the source program whose ⟨File Name⟩ is given and inserts it at the ⟨Start Address⟩ in the ⟨Insert⟩ if present, otherwise in sequence. A ⟨Start Address⟩ occurring within the body of the inserted program will act as though it were in the main program file. A ⟨Start Address⟩ in the ⟨Program Name Line⟩ of the inserted program is ignored. The inserted program takes the multifile ID name from the program being translated. For example:

BCDADD/AFORCE may be inserted into a program named DECVAD/AFORCE. There may be seven levels of nesting. A label may be redefined in an inserted subprogram. An inserted program may reference a label in the program which requested

it provided the label has not (yet) been defined locally. The most local current definition of a label is used. If labels are not defined during a subprogram the translator assumes they are at a more global level. Labels referenced but never defined result in a warning list of undeclared labels. Caution: Forward jumps within a subprogram to a label that already exists globally will use the global label value. Upon completion of an ⟨Insert⟩ of a subprogram, labels defined in that inserted subprogram disappear. A subsequent backward jump or use of a label constant will use the global value, even though the same label was defined in the subprogram.

Each instruction results in a microprogram word. Any instruction may be labeled as a symbolic reference for control transfer. Although transfer to a ⟨Literal Assignment⟩ is permitted it should be used with caution.

Comments - In order to include explanatory material at various points in a program, two conventions exist as defined.

1. COMMENT { any sequence of comment characters except ";" };

   The comment statement acts the same as a ";" and may appear anywhere a ";" may occur if within a line of program. As multi-line documentation the ";" terminator indicates that the micro-translator should resume processing code. Always follow a comment statement with a ";".

2. % { any sequence of comment characters until end of line }

   All comment characters after the % in a line of program are ignored by the microtranslator.

Comments are for documentation purposes only. They appear only in the source file, are significant only in listings and do not affect the machine language generated.

Example

```
PROGRAM READIT
Device *3
SANDY:  Device = LIT; COMP 13 = SAR % LIT = 3 and SAR = 19
        LIT L = BR1
        DL1; A1 + B001 = A1
        INSERT TESTLK
```

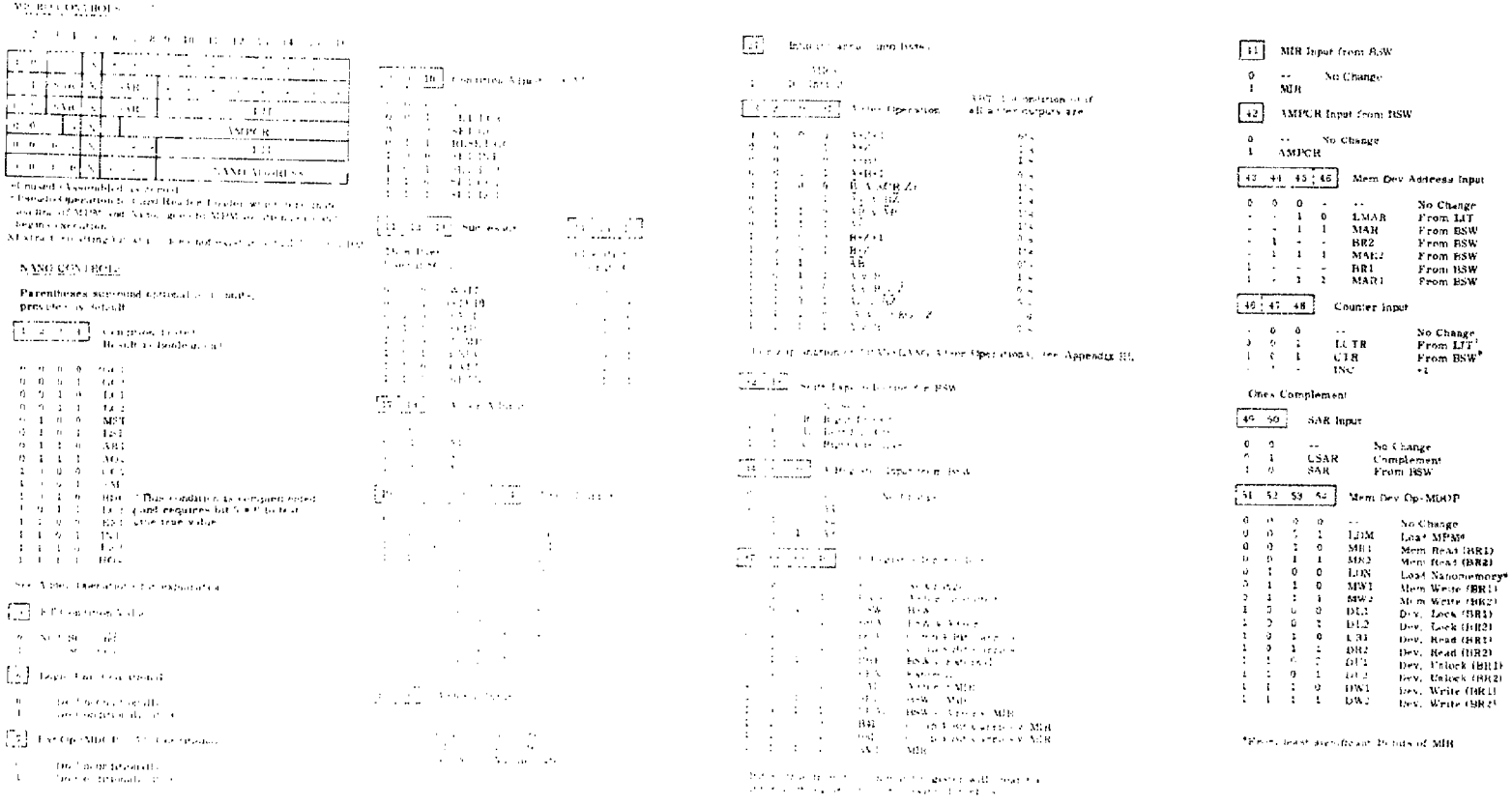


Figure 49. Detailed Nanobit Assignments

```
COMMENT  The routine TESTLK tests to see if device is
         locked to Interpreter.
SANDY - 1 = AMPCR
JUMP;
END;
```

MICROPROGRAMMING EXAMPLES

The Interpreter microprogramming reference card (Figure 49) specifies the use of each of the MPM and Nano bits and defines the meaning of the mnemonics found in the microprogram examples.

Three simple examples demonstrating the microprogramming of the Interpeter are shown: in Figure 50 - Binary Multiply, Figure 51 - Fibonacci Series Generation and Figure 53 - "S" Memory to Micromemory and Nanomemory Loader (S to M Loader). The comments serve to explain the function of each microinstruction step. Figure 52 shows the microtranslator output (1 and 0 patterns for MPM and Nano) for the Binary Multiply example. The S to M Loader is described in more detail in the next section.

Assumptions

(1) Sign-magnitude number representation

(2) Multiplier in A3; multiplicand in B

(3) Double length product required with resulting most significant part, with sign, in B and least significant part in A3

1. A3 XOR B→ ;if LC1

2. B_{0TT}→ A2; if MST then Set LC1

   Comment: Step 1 resets LC1. Steps 1 and conditional part of 2 check signs; if different, LC1 is set.

3. B_{000}→B, LCTR

   Comment: Steps 2 and 3 transfer multiplicand (0 sign) to A2 and clear B.

4. "N"→ LIT; 1→ SAR

   Comment: Steps 3 and 4 load the counter with the number (N = magnitude length) to be used in terminating the multiply loop and load the shift amount register with 1.

5. A3 R→A3; Save

   Comment: Begins test at least bit of multiplier and sets up loop.

6. LOOP: If not LST then B_{0TT}C→B skip else step

7. A2 + B_{0TT}C→B

8. A3 OR B_{T00}R→A3, INC; if not COV then jump else step

   Comment: 6 through 8 - inner loop of multiply (average 2.5 clocks/bit).

9. If not LC1 then B_{0TT} →B; skip else step

10. B_{1TT}→B

    Comment: If LC1 = 0, the signs were the same, hence force sign bit of result in B to be a 0.

11. END

Figure 50. Binary Multiply

Assumptions:

A1 contains starting address for storing of series

A2 contains the number representing the length of the series to be computed

1. A1 ⟶ MAR1

   Comment: Load starting address of series into address register

2. B_{000} ⟶ B, MIR

3. B_{001} ⟶ A3; MW1

   Comment: Load initial element of series (0) into A3 and MIR and write it into starting address. Load second element of series (1) into B.

4. A2 ⟶ CTR;SAVE

   Comment: Load counter with length of series; the counter will be incremented for each generation of an element of the series; COV will signify completion. The SAVE sets up the loop.

5. LOOP: If SAI then A1 + 1 ⟶ A1, MAR1, INC, Step else Wait

   Comment: Set up the next address and increment counter

6. A3 + B ⟶ MIR

   Comment: Generate new element in series and place in MIR

7. B ⟶ A3; BMI, MW1; If NOT COV then Jump else Step

   Comment: Write new element into next address
   Transfer i − 1 element to A3
   Transfer i element to B
   Test counter overflow for completion (go to LOOP, if not done)

8. END

Figure 51. Generation of Fibonacci Series

```
000    PROGRAM BIMULT;
100    A3 XOR  B = : ; IF  LC1;
200    B0TT = : A2; IF MST THEN SET LC1;
300    B000 = : B, LCTR;
400    N = : LIT; 1 = : SAR;
500    A3  R = : A3; SAVE;
600    LOOP:  IF NOT LST THEN B0TT  C = : B; SKIP ELSE STEP;
700    A2 + B0TT C = : B;
800    A3 OR BT00  R = : A3, INC; IF NOT COV THEN JUMP ELSE STEP;
900    IF NOT LC1 THEN B0TT = : B; SKIP ELSE STEP;
1000   B1TT = : B;
1100   END
```

```
0          NANO ADDRESS=    0          0000   00000000000
     3     5    13   16   17   18   19   21   23   29   30
1          NANO ADDRESS=    1          0000   00000000001
     2     5    7    8    9    10   13   16   21   23   30   35
2          NANO ADDRESS=    2          0000   00000000010
     13    16   30   39   48
3          SAR= 1              LIT = 0      01   0000100000000
4          NANO ADDRESS=    3          0000   00000000011
     12    15   17   18   30   33   36
5          NANO ADDRESS=    4          0000   00000000100
     2     4    6    12   13   16   21   23   30   32   33   39
6          NANO ADDRESS=    5          0000   00000000101
     13    16   17   21   23   30   32   33   39
7          NANO ADDRESS=    6          0000   00000000110
     1     11   16   17   18   19   28   30   31   33   36   47
8          NANO ADDRESS=    7          0000   00000000111
     3     6    12   13   16   21   23   30   39
9          NANO ADDRESS=    8          0000   00000001000
     13    16   19   20   21   23   30   39
```

Figure 52. Microtranslator Output

```
PROGRAM STOMLD
OFFSET  =  20          % OFFSET BETWEEN PRIME AND ALTERNATE COPY
    %    LOAD MPM FROM "S" - AVIONICS SYSTEM ----------------------
    % ------  A3: 4-15: LAST AMPCR;   16-31: MEM ADDR;  32: HALF WD
    % ------  A2: 1-16: START ADR;   23-32: PRES AMPCR VALUE
    % ------  LOAD A2 AND A3 FROM OVERLAY TABLE  (LIT VALUE)      ASM-0
    % ------  BR2:CODE AREA  BR1: PWA OF TASK                     ASM-0
STOM: B L= A3%%                                                   ASM-2
    COMP 1=SAR %                                                  ASM-3
    % -----   A3 NOW LOADED -=-                                   ASM-4
    A3 L =: A2,          %          CLEAR A2 12 LST (AMPCR)       ASM-5
    17 =: SAR ; OVER-1=LIT %                                      ASM-6
    A2 ADD LIT = A2,AMPCR %                                       ASM-7A
SMLOOP: A3 R =: BR2, MAR, B        %     LOAD AMPCR               ASM-8
    1 =: SAR; 3 =: LIT %                 3 =: CTR FOR NANO        ASM-9
    MR2;LCTR; IF RDC %                   READ NEXT MEM HALF-WD    ASM-10
    IF RDC THEN B111 =:, BEX; SKIP ELSE WAIT %                    ASM-11
    BMFAIL -1 =: AMPCR %                                          ASM-12
    IF ABT THEN A3 =: ELSE JUMP % TEST FOR HALF WORD              ASM-13
    IF NOT LST THEN B R =: B ELSE SKIP %                          ASM-14
    16 =: SAR %                                                   ASM-15
    B =: MIR,LCTR %                                               ASM-16
    LDM; EXEC % LOAD MICRO                                        ASM-17
    B R =: B %                                                    ASM-18
    11 =: SAR, 31 =: LIT %                                        ASM-19
    B AND LIT =: B %               TEST  FOR FOLLOWING NANO       ASM-20
    BFFF =: %                                                     ASM-21
    LDNANO -1 =: AMPCR %                                          ASM-22
    IF NOT ABT THEN B EQV LIT = B ELSE JUMP %                     ASM-23
    16=LIT  %     %                                               ASM-24
    STEP        %    %             TEST FOR "DONE"               ASM-25
    B %                                                           ASM-26
    0       =: AMPCR %             JUMP TO "1" IF DONE            ASM-28
    IF NOT ABT THEN A3 + B001 =: A3 ELSE JUMP %                   ASM-29
    SMLOOP -1 =: AMPCR %                                          ASM-30
    A2 + B001 =: A2, AMPCR; JUMP %                                ASM-31
LDNANO: A2 =: AMPCR %                                             ASM-32
    A3   +   B001 =: A3 %                                         ASM-33
    A3 R =: B, BR2, MAR %                                         ASM-34
    1 =: SAR %                                                    ASM-35
    MR2: IF RDC %                                                 ASM-36
    IF RDC THEN B111 =:, BEX, SKIP ELSE WAIT %                    ASM-37
    BMFAIL -1 =: AMPCR %                                          ASM-38
    IF ABT THEN A3 =: ELSE JUMP %                                 ASM-39
    IF NOT LST THEN B R =: B ELSE SKIP %                          ASM-40
    16 =: SAR %                                                   ASM-41
    CTR R =: MAR %                                                ASM-42
    COMP 9 =: SAR %                                               ASM-43
    B =: MIR, INC %                                               ASM-44
    LDN; EXEC  % LOAD NANO                                        ASM-45
    LDNANO -1 =: AMPCR %                                          ASM-46
    IF NOT COV THEN  JUMP %                                       ASM-47
    SMLOOP - 1 =: AMPCR %                                         ASM-48
    A3 ADD B001=A3 %      %                                       ASM-49*
    A2 ADD B001 = A2,AMPCR;JUMP%                                  ASM-49A
BMFAIL:  A2 R=B,          %  SHIFT OFF MAR PART                   ASM-50
16=SAR;OFFSET=LIT         %  AND HALF WORD COUNT                   ASM-51
STOM-1=AMPCR              %                                       ASM-52
B ADD LIT  =   B; JUMP%                                           ASM-53
OVER:
```

Figure 53. S to M Loader

SECTION VIII

MULTIPROCESSING CONTROL PROGRAM AND DEMONSTRATION PROGRAMS

CONTROL PROGRAM

The control program for Multi-Interpreter-Systems is a simple yet comprehensive operating system which is characterized by the following capabilities:

1. Multiprocessing
2. Error recovery

In previous multiprocessing systems, I/O functions and data processing functions have been performed in physically different hardware modules, I/O modules for the former and processor modules for the latter. In the Multi-Interpreter System, however, I/O control and processing functions are all performed by identical Interpreters, and any Interpreter can perform any function simply by a reloading of its microprogram memory. Thus input/output operations become tasks which are indistinguishable to the control program from data processing tasks except that they may require the possession of an I/O device before they can begin to run. Whenever an Interpreter is available it looks through the scheduling cards and runs a task, which may be an I/O task, a processing task, or a task which combines both processing and I/O functions.

The control program includes an automatic error detection and recovery capability. All data is stored redundantly to ensure no loss of data should a failure occur. The control program maintains this redundancy, and does so in such a way that each task may be restarted should a failure occur while it is running.

The plans for the development of a full scale operating system for the Aerospace Multiprocessor are described in U. S. A. F. Avionics Laboratory Technical Report AFAL-TR-72-144 (April, 1972), Aerospace Multiprocessor Executive by Sandra Zucker. A building block technique was developed for this software architecture in order to accommodate the requirements for changing computer activities as well as changing hardware modules. The system software was divided into functional modules that could be linked into a system after each module had been independently validated. Descriptions of the executive modules defining scheduling, resource allocation, error recovery and detection, reconfiguration, and file handling are included in the report.

The control program delivered with the aerospace multiprocessor is a quick, efficient, and easy to debug, method of demonstrating the multiprocessor. It is not a fully automatic operating system with complex functions such as the one described in the report referenced above.

System Loading

Initially, the tasks in the system are allocated fixed program areas in S memory which are loaded from cards by the Program to "S" loader. (A description of the program to "S" loader is given later in this section.) All input to the system is loaded redundantly for error recovery purposes. The programs include a method for detection and recovery from memory and Interpreter failures.

The location in S memory of the microcode for each of the demonstration tasks written for the aerospace multiprocessor and the location of the alternate copy of the microcode for these tasks is shown below.

Program	Location of Microcode	Alternate Microcode	Location in System Table
Plot	0300	3300	(00)02
Program to S load	0E00	3E00	(00)03
Mortgage	2000	5100	(00)04
Sort	0600	3600	(00)05
Matrix multiply	2500	5500	(00)08
Matrix print	2800	5800	(00)0A
Memory dump	1000	4000	(00)0C
Control program	0B00	3B00	---

A system task table is developed in segment 00 (segment is 256 words) which contains an entry for each task available to the system. This entry contains the time by which a running task must be completed before the system decides there is an error. An alternate copy of the system table is developed in segment 30 for error recovery purposes. This alternate copy is updated as the primary copy of the system table is changed.

After the tasks are loaded into S memory, each Interpreter's microprogram and nanomemories are then loaded with the control program microcode (See Figure 54, a block diagram of the control program). The control program in an Interpreter initially tries to lock to the card reader. If it does not succeed, some other Interpreter is using the card reader, and it waits until it can lock. Once locked to the card reader, the control program reads the cards which initiate a task and places their contents (eight 4-bit hexidecimal characters) into selected words of S memory as defined by the card format. Each input card contains the hexidecimal characters to be placed in S memory and some contain the address where these characters are to be stored. A card that does not include an address ("0" card) assumes that its hexidecimal input will be stored in the next consecutive address in S memory following the previous input card.

Card Format:	LXXX	AAAA	XXXX	HHHH	HHHH
	0XXX	0000	XXXX	HHHH	HHHH

The "L" card indicates that AAAA contains the hexidecimal address in S memory where the hexidecimal characters HHHH HHHH will be stored. The X characters indicate letters and numbers that are ignored. These may contain anything but an "N".

The "0" card indicates that HHHH HHHH will be stored in the next address in S memory following the previously stored word.

One input card is a control card, specified by an address of 0001, which gives the task number (which is the location in the system table of the task control word) of the selected task as well as the starting address in S memory for the microcode for that task.

The format for the control cards for the demonstration programs written for the aerospace multiprocessor are given below, where T indicates the task number (location of the task entry in the system table) and SS indicates the segment number for the location of the microcode in S memory for that task.

BLOCK DIAGRAM
CONTROL PROGRAM

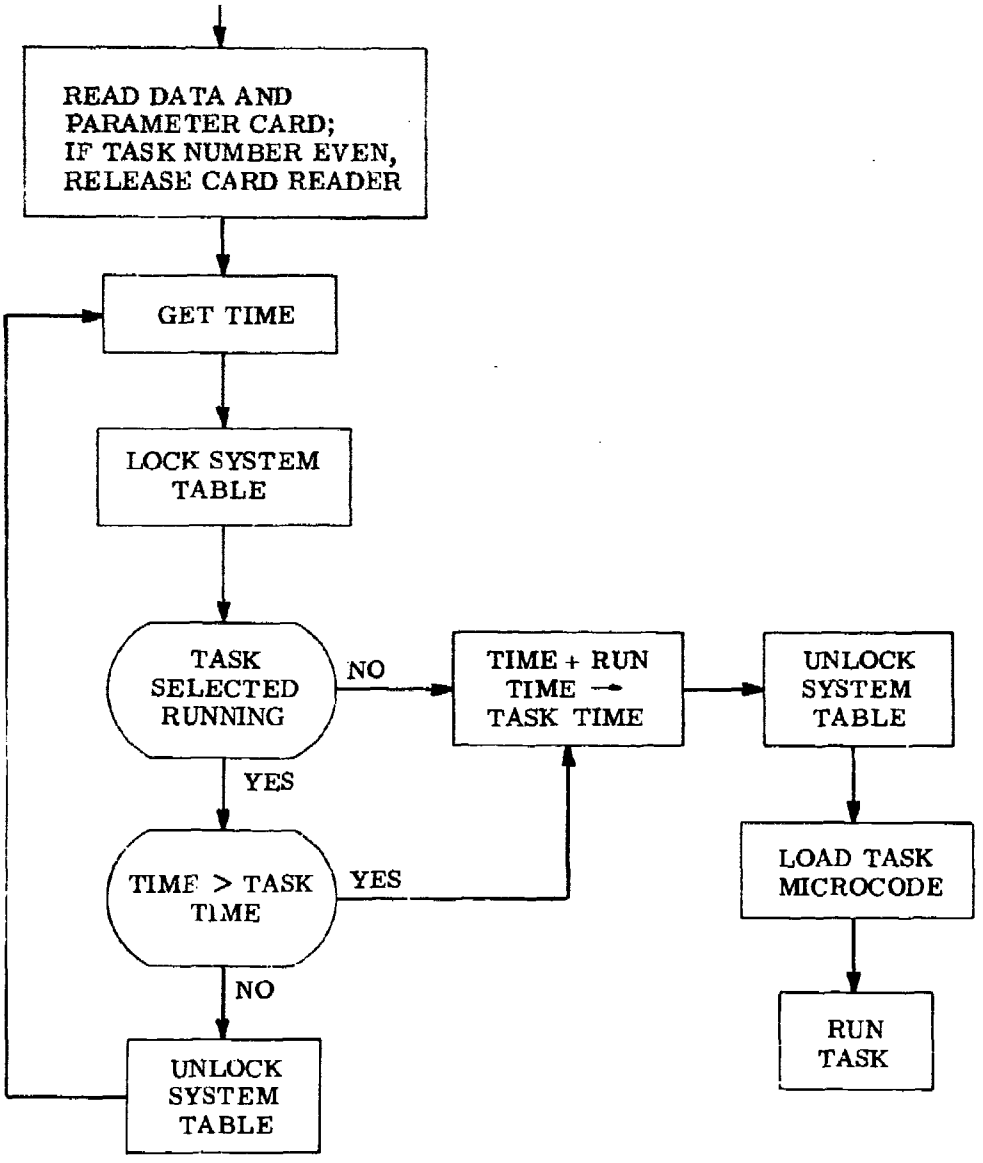


Figure 54. Control Program Flow Diagram

Program	L000	0001	XXXX	T000	SS00
Plot	L000	0001	XXXX	2000	0300
Program to S	L000	0001	XXXX	3000	0E00
Mortgage	L000	0001	XXXX	4000	2000
Sort	L000	0001	XXXX	5000	0600
Matrix multiply	L000	0001	XXXX	8000	2500
Matrix print	L000	0001	XXXX	A000	2800
Memory dump	L000	0001	XXXX	C000	1000

All other input cards are parameter cards for the task and are loaded into a portion of the work area for that task.

An "N" card is the last card that indicates the end of the selection of a single task. The "N" card must contain a single N.

Upon detection of an "N" card the control program stops reading cards and uses location 01 of the system table to get the task number of the selected task. An even task number will cause the card reader to be unlocked, freeing it so that other Interpreters may use it. An odd numbered task requires the card reader in order to read its own data (e.g., sort cards for the sort task), after which the card reader will be unlocked. This contention between Interpreters for use of the card reader and running of tasks is shown in block diagram for the multiprocessor system in Figure 55.

Task Execution and Monitoring

The task number is used to select the task control word from the task table. The task table is locked before a task control word may be examined or changed, by using the global condition bit in the hardware. A task control word of zero defines a task available for running. A non zero task control word implies that another Interpreter is performing the task, or that the task is hung up on another Interpreter.

To check for a task or Interpreter failure, the real time clock is read to obtain the current time. The current time is checked against the time in the task control word which is the upper bound time for the running of the task. If the time in the task control word is less than the time on the real time clock, the task is considered hung and the Interpreter will treat this task as a task available for running.

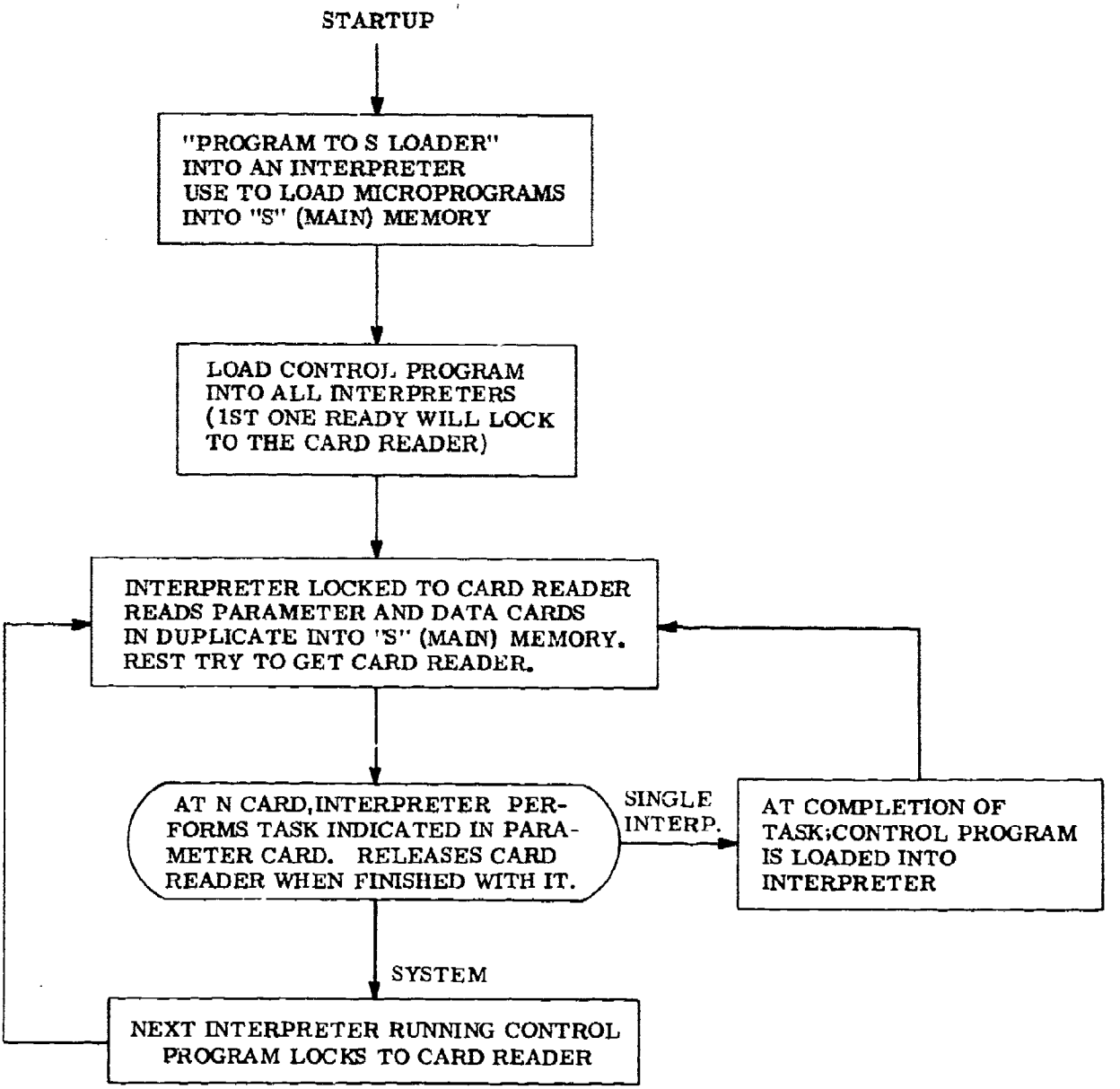


Figure 55. Multiprocessor System Flow Diagram

When a task is still running, and the time on the real time clock is less than the time in the task control word, the global condition bit is reset. Then a new reading is made of the real time clock value. The task control word is again tested after locking the table. This process continues until either the time for running the task elapses or the task is completed by the Interpreter running it.

When a task is available for running, a maximum run time value is added to the time read from the real time clock and the sum is placed into the task control word. The global condition bit is reset (unlocking the table) and the microprogram for the task is read from S memory into the Interpreter's microprogram memory. The task is then executed. A task which uses the card reader (an odd numbered task) must release the card reader as soon as it has completed getting its data.

When a task has been successfully completed by an Interpreter, it resets its task control word to zero and loads the control program from S memory to microprogram memory. To determine the next task, the control program again reads the cards from the card reader.

All information is stored redundantly in S memory. (See memory map in Figure 56.) When a memory failure is detected by an Interpreter, which will affect the running of a task, the Interpreter will reload its own microprogram memory with the alternate S memory program. This program is identical to the prime microprogram except that it uses the alternate work area and data space as input instead of the prime areas.

The detection of a memory failure during the loading of the prime area of a task or the control microprogram will cause the loading of the alternate area of the required program instead. All cards read using the control program will be stored redundantly in S Memory.

S to M Loader

All tasks as well as the control program contain a subroutine (S to M loader) which can load microprogram code from S (main) memory to microprogram memory and to nanomemory. This subroutine (see Figure 57) is bypassed when a task is initiated. When the task is completed or an error is detected, an address is placed in the B register and control is transferred to the S to M loader which loads code into that part of microprogram memory and nanomemory that is not occupied by the S to M loader. When it detects the end code (ONE in the most significant bit of the microinstruction and ZERO in rest of it) it stops reading and jumps to the start of the task just read.

When a task ends, it puts the address of the control program into the B register so that the next task may be selected and executed. If a task has an error, it puts the address of its own alternate copy into the B register for restart. If a task is too large to completely fit into microprogram memory and nanomemory, at the completion of the first or intermediate part of the task, the address of the next part of the task is put into the B register. The task then passes control to the S to M loader subroutine for loading the next task or next part of the same task to be executed. This procedure is shown in Figure 58. The microcode for the S to M loader is shown in Figure 53 of Section VII of this report.

Segment No.	Module 0 (Segments 00-1F)	Module 1 (Segments 20-3F)	Module 2 (Segments 40-5F)
	(Module = 8,192 words; Segment = 256 words)		
0	System Table	Mortgage Work Area	Dump Memory Microcode (Alternate)
1	Plot Work Area	Mortgage Microcode	
2	Sort Work Area		Data Cards
3	Plot Microcode		
4		Matrix Work Area	Input for Sort (Alternate)
5		Matrix Multiply Microcode	
6	Sort Part 1 Microcode		
7			
8	Sort Part 2 Microcode	Matrix Print Microcode	
9			
A			
B	Control Program Microcode	Matrix Print Work Area	
C		Matrix A Data	
D		Matrix B Data	
E	Program to S Loader	Matrix C Data	
F	Microcode		
10	Dump Memory Microcode	Alternate System Table	Alternate Mortgage Work Area
11		Alternate Plot Work Area	Mortgage Microcode (Alternate)
12	Data Cards	Alternate Sort Work Area	
13	Input for Sort	Plot Microcode (Alternate)	
14			Alternate Matrix Work Area
15			Matrix Multiply Microcode (Alternate)
16		Sort Part 1 Microcode (Alternate)	
17			
18		Sort Part 2 Microcode (Alternate)	Matrix Print Microcode (Alternate
19			
1A			
1B		Control Program Microcode (Alternate)	Alternate Matrix Print Work Area
1C			Alternate Matrix A
1D			Alternate Matrix B
1E		Program to S Loader Microcode (Alternate)	Alternate Matrix C
1F			

Figure 56. Memory Map

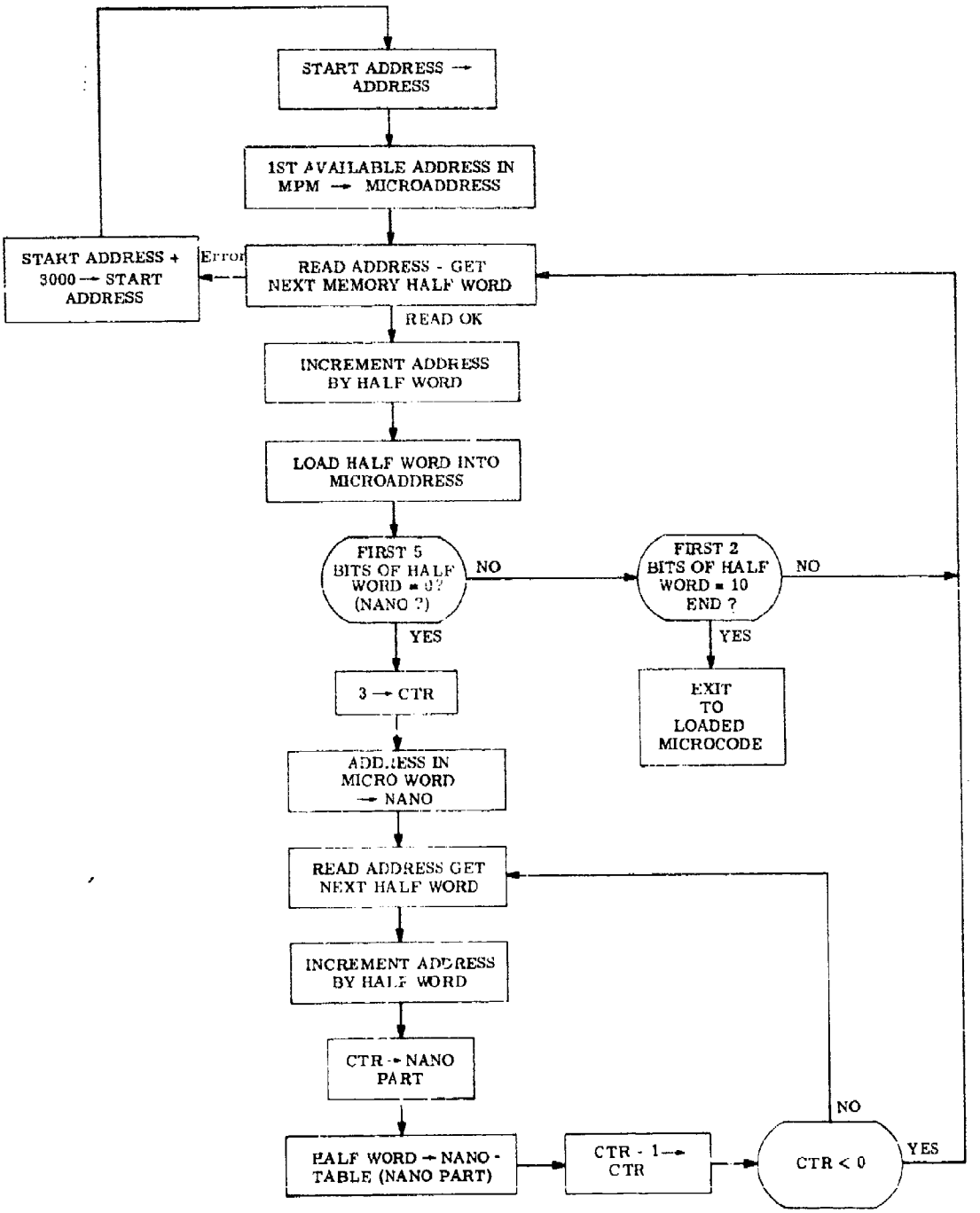


Figure 57. Load Microprogram Memory from Main Memory Flow Diagram

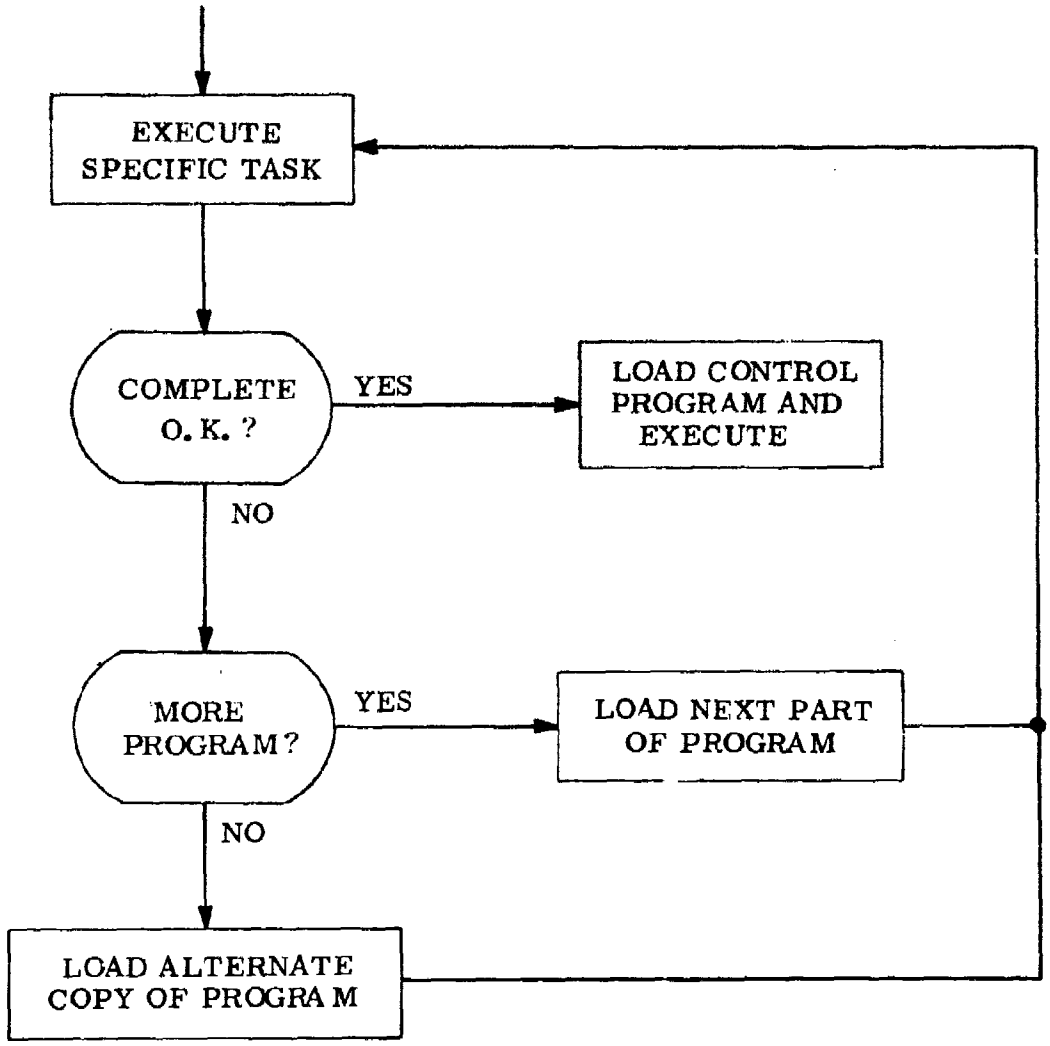


Figure 58. Task Control Flow Diagram

DEMONSTRATION PROGRAMS

All the demonstration programs are microprogrammed and are loaded from S memory into microprogram and nanomemory in order to be executed. They are like a single large instruction on a conventional machine. Therefore no interpretation of S memory instructions is necessary in this demonstration.

The demonstration programs were written to be indicative of a specific type of application as indicated below.

Problem	Type Application
Plot	Graphic Display Table Lookup
Mortgage	Table Building Simple Arithmetic
Sort	Data Manipulation Data Processing
Matrix	Arithmetic Operations (Many Multiplies)
Dump	Debugging Aid
Program to "S" loader	Loading S Memory

All the demonstration tasks which use data and parameters contain a work area segment. This work area allows for the storing of parameters. temporary work space, buffers and pointers to data or program areas used by the task. Thus, the work area for the matrix routine contains pointers to the three matrix areas as well as the parameters i, j, and k. Changing any of these parameters or pointers will change what is executed by the task. The locations of the parameters within the work area for all demonstration programs are shown in Table V.

Memory Dump

The Memory Dump routine prints all the contents of S memory without changing or disturbing any of the memory locations. Each 32-bit word in S memory is printed in a format of eight 4-bit hexidecimal characters. The words are grouped into an address followed by eight words of memory and then printed as a line. If a line is identical to the previous eight words printed then it is omitted. The memory dump is a debugging aid used to detect changes in memory. An example of the output from a memory dump appears in Figure 59.

Table V. Demonstration program parameters

Program	Location of Parameter	Parameter Description and Example	Parameter Card Format				
Plot	0114	Starting angle 0^{o}	L000	0114	0000	0000	0000
	0115	Ending angle 360^{o}	0000	0000	0000	0000	0168
	0116	Delta (degrees between pts) 1^{o}	0000	0000	0000	0000	0001
Mortgage	2014	Principal $22,500.00	L000	2014	0000	0225	0000
	2015	Rate 8.50%	0000	0000	0000	0000	0850
	2016	Payment $ 250.00	0000	0000	0000	0002	5000
Sort	020A	Read card deck Yes = 0 No = 1	L000	020A	0000	0000	0000
	020C	Segment loc. of cards in mem 12	L000	020C	0000	0000	0012
	0214	Number of char in key 20	L000	0214	0000	0000	0014
	0215	Starting character of key 10	0000	0000	0000	0000	000A
	0217	Direction of sort descending = 0	L000	0217	0000	0000	0000
		ascending = 1	L000	0217	0000	0000	0001
Matrix multiply	240C	Location matrix A 2C	L000	240C	0000	0000	002C
Matrix print (24 changed to 2B)	240D	Location matrix B 2D	0000	0000	0000	0000	002D
	240E	Location matrix C 2E	0000	0000	0000	0000	002E
	2416	J (index) 8	L000	2416	0000	0000	0008
	2417	I (index) 10	0000	0000	0000	0000	000A
	2418	K (index) 12	0000	0000	0000	0000	000C

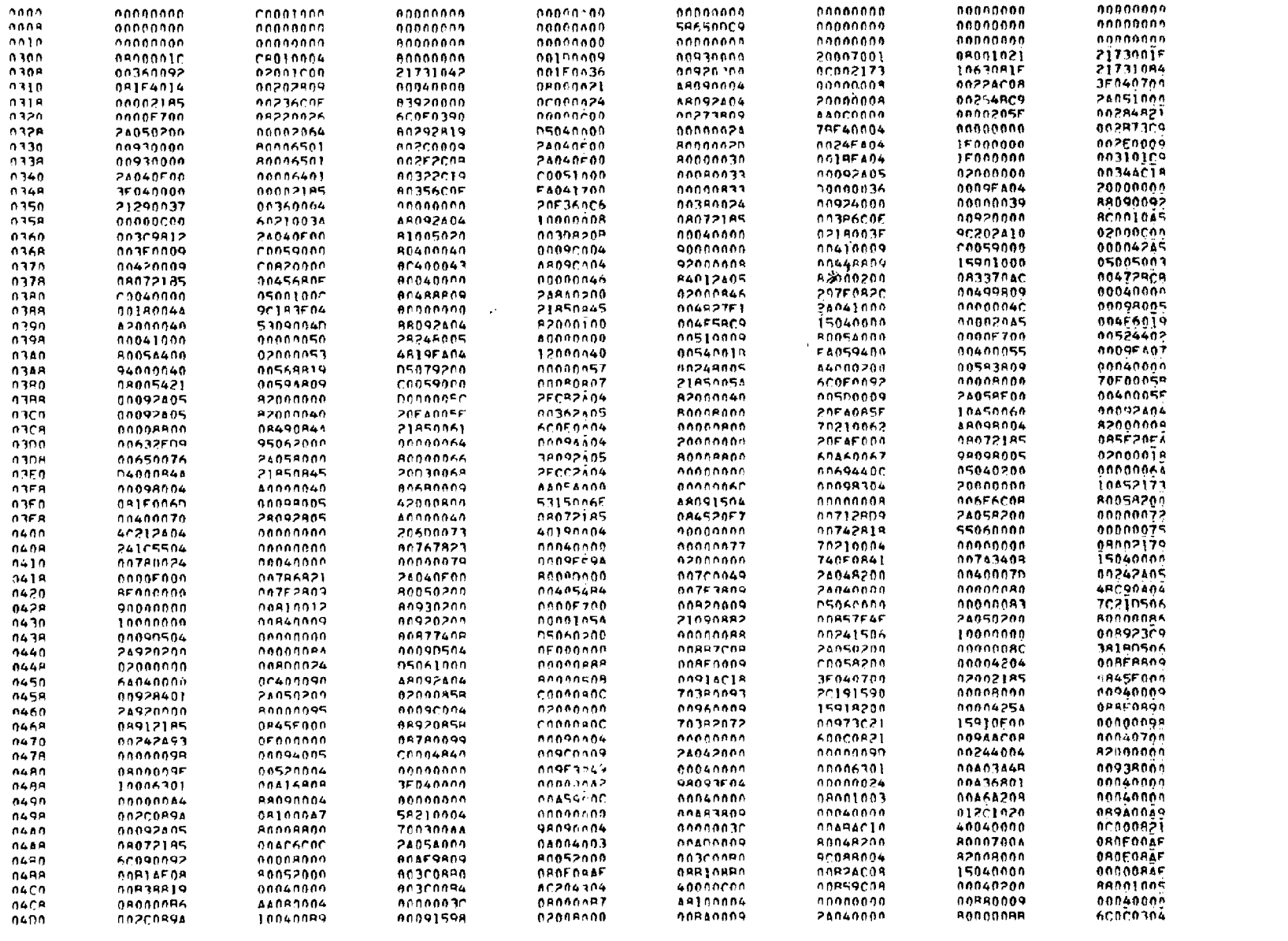


Figure 59. Example of Memory Dump Output

Program to "S" Loader

The Program to "S" loader reads cards from the card reader in a format generated by the Translator and places them into S memory. An L card precedes the program cards for each microprogram to be loaded to indicate where in S memory each of the microprograms will be loaded, and an R card is used to indicate the end of the Program to "S" Loading function.

L Card L000 AAAA
R000 0000

where AAAA = starting address in S memory for the microprogram

Each microinstruction is stored into 16 bits of memory. If a microinstruction points to a nanoinstruction which is used for the first time, it will be stored following the micro in the next 64 bits of memory. All the micro's and nano's are packed in S memory into 32-bit words. Nanos that are used repeatedly need be stored in S memory only once.

Microinstruction format

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	*
1	1	SAR		-	SAR										
0	1	SAR		-	SAR			LIT							
0	0	1	-	-	-	AMPCR									
0	0	0	1	-	-	-	-	LIT							
0	0	0	0	0	-	-	-	NANO ADDRESS							
0	0	0	0	1	-	-	-	NANO ADDRESS							

All instructions except a type 1 instruction ignore bit 5. Type 1 instructions use bit 5 to determine whether a nano must be loaded (bit 5 = 0) in the nano table or if it has been used already by a previously defined microinstruction (bit 5 = 1). A - indicates the bit can be either a 1 or 0 since it is ignored by the loader.

Plot

The plot routine plots the sine curve using (*) and cosine curve using (@) on the printer. The y axis is horizontal (since the size is fixed) and the x axis is vertical. Each line is printed with the angle in degrees defining the line on the left and the symbol of the sine and cosine plots (* and @) in its proper position along the y axis. The user can specify the starting angle (in degrees), the ending angle, and a delta (increment in degrees between points to be plotted).

*This is a pseudo-instruction which is used to indicate the end of a program.

Starting angle	L000	0114	0000	0000	AAAA
Ending angle	L000	0115	0000	0000	AAAA
Delta	L000	0116	0000	0000	0DDD

AAAA = angle in hexidecimal (002D = 45°)

DDD = increment in angle for each print line in hexidecimal (000F = 15°)

An example of the plot output appears in Figure 60.

Mortgage

The mortgage program produces a mortgage table which gives a list of the monthly payments of a mortgage and the results of each payment. This includes the pay period number, the amount of interest paid this payment, the amount of this payment used for amortization, the remaining principal, the accumulated interest, and the number of years of payment. The user must supply the principal, the monthly payments and yearly rate as input. These parameters are entered into the task work area via the control program.

Principal	L000	2014	0000	PPPP	PPPP
Rate	L000	2015	0000	0000	RRRR
Payment	L000	2016	0000	0MMM	MMMM

PPPPPPPP = principal in 4-bit decimal digits (02250000 = $22,500.00)

RRRR = yearly rate in 4-bit decimal digits (0850 = 8.50%)

MMMMMMMM = monthly payments in 4-bit decimal digits (0025000 = $250.00)

An example of the mortgage output appears in Figure 61.

Sort

The Sort routine reads a deck of cards and sorts them according to the starting character and length of a key defined by the user in the work area. The sort may be either an ascending or descending sort depending on a parameter. The same deck of cards may be sorted using different keys and in different directions with-

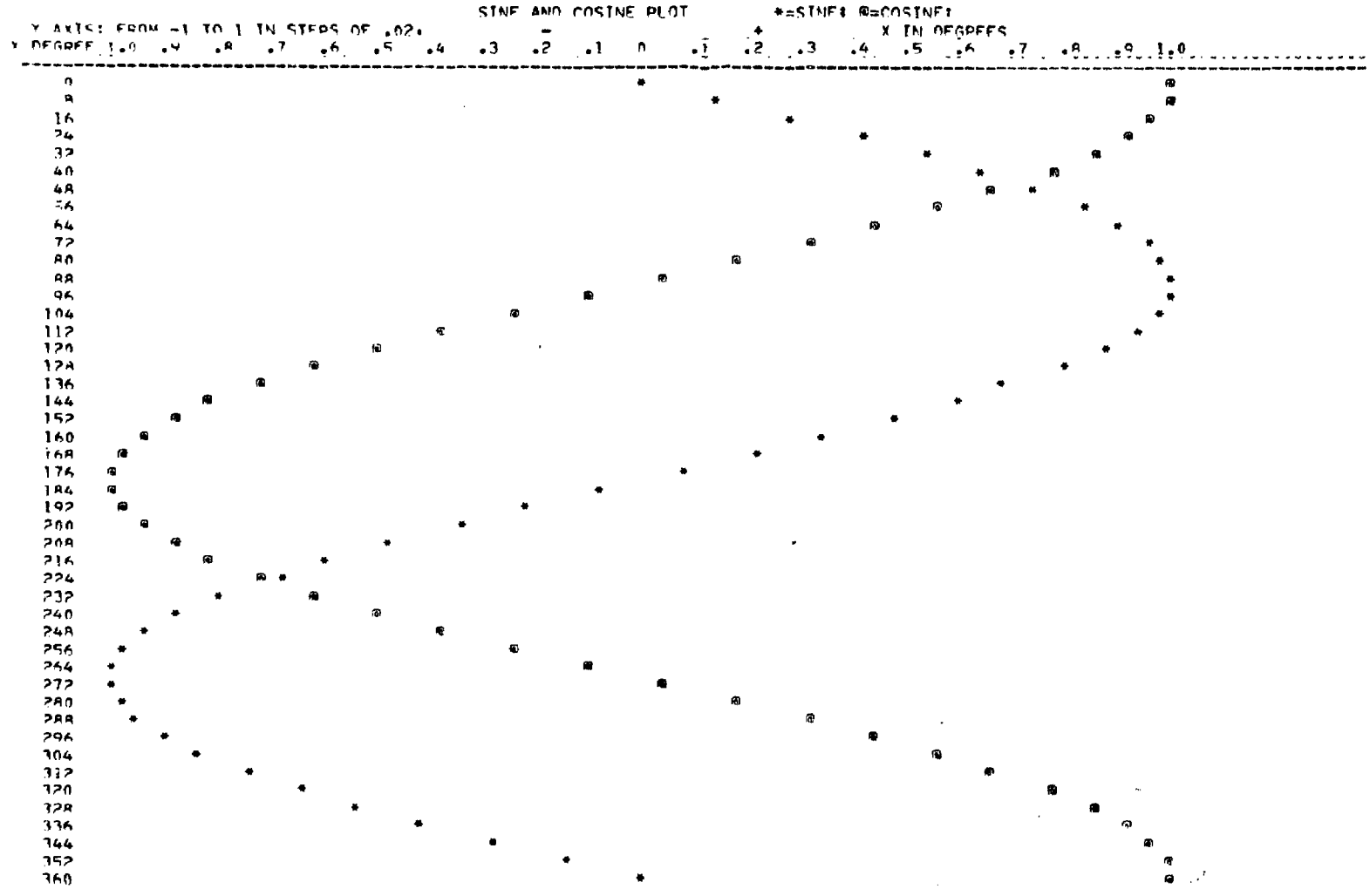


Figure 60. Example of Plot Routine Output

MORTGAGE

PRINCIPAL = $ 25,000.00 RATE 7.00 PAYMENTS = 375.00

PERIOD	INTEREST	AMORTIZATION	PRINCIPAL	ACCUMULATED INTEREST	YEAR
1	$ 145.00	$ 230.00	$ 24,770.00	$ 145.00	1
2	$ 143.67	$ 231.33	$ 24,538.67	$ 288.67	1
3	$ 142.32	$ 232.68	$ 24,305.99	$ 430.99	1
4	$ 140.97	$ 234.03	$ 24,071.96	$ 571.96	1
5	$ 139.62	$ 235.38	$ 23,836.58	$ 711.58	1
6	$ 138.25	$ 236.75	$ 23,599.83	$ 849.83	1
7	$ 136.88	$ 238.12	$ 23,361.71	$ 986.71	1
8	$ 135.50	$ 239.50	$ 23,122.21	$ 1,122.21	1
9	$ 134.11	$ 240.89	$ 22,881.32	$ 1,256.32	1
10	$ 132.71	$ 242.29	$ 22,639.03	$ 1,389.03	1
11	$ 131.31	$ 243.69	$ 22,395.34	$ 1,520.34	1
12	$ 129.89	$ 245.11	$ 22,150.23	$ 1,650.23	1
13	$ 128.47	$ 246.53	$ 21,903.70	$ 1,778.70	2
14	$ 127.04	$ 247.96	$ 21,655.74	$ 1,905.74	2
15	$ 125.60	$ 249.40	$ 21,406.34	$ 2,031.34	2
16	$ 124.16	$ 250.84	$ 21,155.50	$ 2,155.50	2
17	$ 122.70	$ 252.30	$ 20,903.20	$ 2,278.20	2
18	$ 121.24	$ 253.76	$ 20,649.44	$ 2,399.44	2
19	$ 119.77	$ 255.23	$ 20,394.21	$ 2,519.21	2
20	$ 118.29	$ 256.71	$ 20,137.50	$ 2,637.50	2
21	$ 116.80	$ 258.20	$ 19,879.30	$ 2,754.30	2
22	$ 115.30	$ 259.70	$ 19,619.60	$ 2,869.60	2
23	$ 113.79	$ 261.21	$ 19,358.39	$ 2,983.39	2
24	$ 112.28	$ 262.72	$ 19,095.67	$ 3,095.67	2
25	$ 110.75	$ 264.25	$ 18,831.42	$ 3,206.42	3
26	$ 109.22	$ 265.78	$ 18,565.64	$ 3,315.64	3
27	$ 107.68	$ 267.32	$ 18,298.32	$ 3,423.32	3
28	$ 106.13	$ 268.87	$ 18,029.45	$ 3,529.45	3
29	$ 104.57	$ 270.43	$ 17,759.02	$ 3,634.02	3
30	$ 103.00	$ 272.00	$ 17,487.02	$ 3,737.02	3
31	$ 101.42	$ 273.58	$ 17,213.44	$ 3,838.44	3
32	$ 99.84	$ 275.16	$ 16,938.28	$ 3,938.28	3
33	$ 98.24	$ 276.76	$ 16,661.52	$ 4,036.52	3
34	$ 96.64	$ 278.36	$ 16,383.16	$ 4,133.16	3
35	$ 95.02	$ 279.98	$ 16,103.18	$ 4,228.18	3
36	$ 93.40	$ 281.60	$ 15,821.58	$ 4,321.58	3
37	$ 91.76	$ 283.24	$ 15,538.34	$ 4,413.34	4
38	$ 90.12	$ 284.88	$ 15,253.46	$ 4,503.46	4
39	$ 88.47	$ 286.53	$ 14,966.93	$ 4,591.93	4
40	$ 86.81	$ 288.19	$ 14,678.74	$ 4,678.74	4
41	$ 85.14	$ 289.86	$ 14,388.88	$ 4,763.88	4
42	$ 83.45	$ 291.55	$ 14,097.33	$ 4,847.33	4
43	$ 81.76	$ 293.24	$ 13,804.09	$ 4,929.09	4
44	$ 80.06	$ 294.94	$ 13,509.15	$ 5,009.15	4
45	$ 78.35	$ 296.65	$ 13,212.50	$ 5,087.50	4
46	$ 76.63	$ 298.37	$ 12,914.13	$ 5,164.13	4
47	$ 74.90	$ 300.10	$ 12,614.03	$ 5,239.03	4
48	$ 73.16	$ 301.84	$ 12,312.19	$ 5,312.19	4
49	$ 71.41	$ 303.59	$ 12,008.60	$ 5,383.60	5
50	$ 69.65	$ 305.35	$ 11,703.25	$ 5,453.25	5
51	$ 67.88	$ 307.12	$ 11,396.13	$ 5,521.13	5
52	$ 66.10	$ 308.90	$ 11,087.23	$ 5,587.23	5
53	$ 64.30	$ 310.70	$ 10,776.53	$ 5,651.53	5
54	$ 62.50	$ 312.50	$ 10,464.03	$ 5,714.03	5
55	$ 60.69	$ 314.31	$ 10,149.72	$ 5,774.72	5
56	$ 58.87	$ 316.13	$ 9,833.59	$ 5,833.59	5
57	$ 57.03	$ 317.97	$ 9,515.62	$ 5,890.62	5
58	$ 55.19	$ 319.81	$ 9,195.81	$ 5,945.81	5
59	$ 53.33	$ 321.67	$ 8,874.14	$ 5,999.14	5
60	$ 51.47	$ 323.53	$ 8,550.61	$ 6,050.61	5
61	$ 49.59	$ 325.41	$ 8,225.20	$ 6,100.20	6
62	$ 47.71	$ 327.29	$ 7,897.91	$ 6,147.91	6
63	$ 45.81	$ 329.19	$ 7,568.72	$ 6,193.72	6
64	$ 43.90	$ 331.10	$ 7,237.62	$ 6,237.62	6
65	$ 41.98	$ 333.02	$ 6,904.60	$ 6,279.60	6
66	$ 40.05	$ 334.95	$ 6,569.65	$ 6,319.65	6
67	$ 38.10	$ 336.90	$ 6,232.75	$ 6,357.75	6
68	$ 36.15	$ 338.85	$ 5,893.90	$ 6,393.90	6
69	$ 34.18	$ 340.82	$ 5,553.08	$ 6,428.08	6
70	$ 32.21	$ 342.79	$ 5,210.29	$ 6,460.29	6
71	$ 30.22	$ 344.78	$ 4,865.51	$ 6,490.51	6
72	$ 28.22	$ 346.78	$ 4,518.73	$ 6,518.73	6
73	$ 26.21	$ 348.79	$ 4,169.94	$ 6,544.94	7
74	$ 24.18	$ 350.82	$ 3,819.12	$ 6,569.12	7
75	$ 22.15	$ 352.85	$ 3,466.27	$ 6,591.27	7
76	$ 20.10	$ 354.90	$ 3,111.37	$ 6,611.37	7
77	$ 18.04	$ 356.96	$ 2,754.41	$ 6,629.41	7
78	$ 15.97	$ 359.03	$ 2,395.38	$ 6,645.38	7
79	$ 13.89	$ 361.11	$ 2,034.27	$ 6,659.27	7
80	$ 11.80	$ 363.20	$ 1,671.07	$ 6,671.07	7
81	$ 9.69	$ 365.31	$ 1,305.76	$ 6,680.76	7
82	$ 7.57	$ 367.43	$ 938.33	$ 6,688.33	7
83	$ 5.44	$ 369.56	$ 568.77	$ 6,693.77	7
84	$ 3.30	$ 371.70	$ 197.07	$ 6,697.07	7
85	$ 1.14	$ 197.07	$ .00	$ 6,698.21	8

Figure 61. Example of Mortgage Table Output

out reading the deck in each time. The results of each sort will be printed giving the original position of the card in the deck.

Read new set of cards	L000	020A	0000	0000	0000
Use old set of cards	L000	020A	0000	0000	0001
Pointer to sort cards	L000	020C	0000	0000	00YY
Number of characters in key	L000	0214	0000	0000	00KK
Starting character in key	L000	0215	0000	0000	00SS
Direction of sort descending	L000	0217	0000	0000	0000
Direction of sort ascending	L000	0217	0000	0000	0001

YY = segment number for storage of cards to be sorted

KK = number of characters in sort key in hexidecimal (up to 64)

SS = location of starting key in card character of sort

The last card of a deck of cards to be sorted must contain an illegal character (?). An example of the card input to the sort and the several outputs of the sort, using different keys and different sort directions, appear in Figure 62.

Matrix Multiply and Print

The Matrix Multiply program allows for the construction of a matrix which is the product of two given matrices. Each matrix element is an integer (positive or negative). The dimensions of the matrices may vary and will be defined by parameters stored in the work area. Pointers to the input matrices and to the storage area for the output matrix will also be stored in the work area.

The Matrix Multiply program has been written so that more than one Interpreter may work on the same matrix at the same time, each performing its own unique set of row calculations. Each of these processes must have its own work area indicating a starting row position and an entry for the number of processors that are performing the multiply.

The matrix print routine must start when the matrix multiply has been completed. This routine will print the input matrices and the resultant matrix on the printer.

The user of the matrix multiply and matrix print procedures must specify parameters of both of these routines. These parameters determine the dimensions and locations of the matrices to be multiplied:

$$A_{ij} \times B_{jk} = C_{i\,k}$$

Pointer to matrix	A	L000	WWOC	0000	0000	00YY
	B	L000	WWOD	0000	0000	00YY
	C	L000	WWOE	0000	0000	00YY
i		L000	WW17	0000	0000	00DD
j		L000	WW16	0000	0000	00DD
k		L000	WW18	0000	0000	00DD

WW = segment number for work area storage of matrix multiply (24) and matrix print (2B) in hexidecimal

YY = segment number for location of matrices in hexidecimal

DD = dimension of matrices in hexidecimal

Maximum size of matrix is 256 (size of segment).

Therefore the maximum dimension size is limited by the following formulas:

$$i \times j \leq 256$$
$$j \times k \leq 256$$
$$i \times k \leq 256$$

Since no more then 16 numbers can fit across the page for the matrix print, the number of elements in a row should be no more than 16.

$$i \text{ and } j \leq 16$$

Two examples of the matrix print output appear in Figure 63.

CONFIDENCE ROUTINES

Four confidence routines, AERO1/KDK, AERO2/KDK, AERO3/KDK, and AERO4/KDK test internal Interpreter functions. These routines must be loaded directly into the microprogram memory and are not run under the control program. The following assumptions are made in the confidence routines:

A RIM B works

No errors in MPM or Nanomemory that do not appear in instruction 1 which is a dummy instruction used to set as many nano bits as possible.

A+0 and 0+B Work.

AERO1/KDK exercises the source-destination functions of the Interpreter, the successor controls, and the condition tests LST, MST, ABT, and AOV. The tests are designed to test from the simple to more complex. The detection of an error in the initial tests will cause a wait-wait at the nearest point to the error. Upon completion of testing of the successor controls all errors will exit to a standard-error routine.

AERO2/KDK exercises the SAR, CTR, and shifting functions. This test may also be considered as a test of the barrel switch. This test assumes that the first test (AERO1/KDK) runs successfully.

AERO3/KDK exercises the adder and carry logic of the Interpreter. This section of the code is divided into two parts. Part 1 exercises both A+B and A+B+1 logic. Part 2 exercises the logic type instructions (NOR, NRI, NAN, XOR, NIM, IMP, EQV, AND, RIM, OR, A+0, 0+B).

A subsection of Part 2 exercises four instructions (OAD, AAD, A-B and A-B-1) that exist in the instruction set on other versions of the Interpreter. This section of code exercises no new functions on the LSI Interpreters.

Corresponding to each section (or subsection) there is a subroutine which performs the final comparison of results. The error indication and reporting for each section is done by calling a standard error routine from the corresponding subroutine.

AERO4/KDK exercises those remaining areas of the Interpreter not tested in the previous tests. This test exercises: LC1, LC2, LC3, INT, GC1, GC2, AOV, IC, CSAR, and B Register inputs: BAD, BBA, BBI, BC4, BC8.

```
COMP SURVEYS   MACDOUGALL  M H      70:03   COMPUTER SYSTEM SIMULATION
ACM COMM       DALEY  R:DENNIS J    68:05   VIRTUAL MEMORY SHARING IN MULTICS
COMP SURVEYS   MCKINNEY  J M       69:06   SURVEY ANALYTICAL TIME SHARING MODELS
ACM COMM       NIELSEN  N R         70:08   ALLOCATION COMPUTER RESOURCES
COMPELL THES   HOLT  R C            71:06   DEADLOCK IN COMPUTER SYSTEMS
ACM COMM       IRONS  E T            70:01   EXPERIENCE WITH EXTENSIBLE LANGUAGE
IBM SYSTEMS    HAVENDER  J W        68:07   AVOIDING DEADLOCK  MULTITASKING SYSTEM
ACM COMM       HABERMAN  A N        69:07   PREVENTION OF SYSTEM DEADLOCK
AFIPS FJCC     DENNING P J           68:09   THRASHING:ITS CAUSES AND PREVENTION
COMP SURVEYS   DENNING P J           70:09   VIRTUAL MEMORY
AL  COMM       DENNIS  J B           68:05   POSITION PAPER COMPUTING COMMUNICATNS
ACM COMM       BERNSTEIN:SHARPE      71:02   POLICY DRIVEN SCHEDULER FOR TSS
COMP SURVEYS   COFFMAN:ELPHICK       71:06   SYSTEM DEADLOCKS
MIT MEMO       DENNIS  J B           69:06   FUTURE TRENDS IN TIME SHARING SYSTEMS
COMP SURVEYS   HOFFMAN  L J          69:06   COMPUTERS AND PRIVACY
IRE TRANS      KILBURN:EDWARDS       62:04   ONE LEVEL STORAGE SYSTEM
AFIPS SJCC     RUBINOFF  L           70:05   CONTINUUM TIME SHARING SCHEDULING
AFIPS FJCC     LAMPSON  B W          69:09   DYNAMIC PROTECTION STRUCTURES
PRINCETON      LAMPSON  B W          71:03   PROTECTION
IEEE COMPUTER  ABATE  J:DUBNER  H    69:11   OPTIMIZING PERFORMANCE DRUM STORAGE
ACM COMM       DIJKSTRA  E W         65:09   SOLUTION IN CON CURRENT PROG CONTROL
ACM COMM       DIJKSTRA  E W         68:05   STRUCTURE THE MULTIPROGRAMMING SYSTEM
ACM COMM       GRAHAM  R M           68:05   PROTECTION INFORMATION PROCESSING
AS SYMP        POOLE  P:WAITE  W     69:10   MACHINE INDEPENDENT SOFTWARE
ACM COMM       RANDELL:KUEHNER       68:05   DYNAMIC STORAGE ALLOCATION SYSTEMS
COMP SURVEYS   ROSIN  S              69:03   ELECTRONIC COMPUTERS:HISTORICAL SURVEY
COMP SURVEYS   ROSIN  R F            69:03   SUPERVISORY AND MONITOR SYSTEMS
ACM COMM       SUTHERLAND  I E       68:06   FUTURES MARKET IN COMPUTER TIME
COMP REVIEWS   TRIMBLE  J JR         68:05   TIME SHARING BIBLIOGRAPHY
ACM COMM       WAITE  W M            70:07   MOBILE PROGRAMMING SYSTEM:STAGE 2
DATAMATION     CORBATO  F J          69:05   PL/1 AS TOOL FOR SYSTEM PROGRAMMING
IEEE INTNATL   CREECH  R A          70:06   IMPLEMENTATION OF OPERATING SYSTEMS
AFIPS FJCC     CRITCHLOW  A J        63:09   GENERALIZED MULTIPROGRAMMING SYSTEMS
ACM COMM       DENNING  P J         68:05   WORKING SET MODEL PROGRAM BEHAVIOR
ACM COMM       WILKES  M V           68:01   COMPUTERS THEN AND NOW
```

(a) Card Input Sequence

Figure 62. Example of Sort Routine Output

COUNT	POSITION		XXXXXXXXXXXXXXXXXXXXXXXXXXXX	SORT	
1	20	IEEE COMPUTER	AMATE J:DUNNER M	69:11	OPTIMIZING PERFORMANCE DRUM STORAGE
2	12	ACM COMM	BERNSTEIN:SHARPE	71:02	POLICY DRIVEN SCHEDULER FOR TSS
3	13	COMP SURVEYS	COFFMAN:ELPHICK	71:06	SYSTEM DEADLOCKS
4	31	DATAMATION	CORMATO F J	69:05	PL/I AS TOOL FOR SYSTEM PROGRAMMING
5	32	IEEE INTNATL	CREECH R A	70:06	IMPLEMENTATION OF OPERATING SYSTEMS
6	33	AFIPS SJCC	CWITCHLOW A J	63:09	GENERALIZED MULTIPROGRAMMING SYSTEMS
7	2	ACM COMM	DALEY R:DENNIS J	68:05	VIRTUAL MEMORY SHARING IN MULTICS
8	34	ACM COMM	DENNING P J	68:05	WORKING SET MODEL PROGRAM BEHAVIOR
9	9	AFIPS FJCC	DENNING P J	68:09	THRASHING:ITS CAUSES AND PREVENTION
10	10	COMP SURVEYS	DENNING P J	70:09	VIRTUAL MEMORY
11	11	ACM COMM	DENNIS J B	68:05	POSITION PAPER COMPUTING COMMUNICATNS
12	14	MIT MEMO	DENNIS J H	69:05	FUTURE TRENDS IN TIME SHARING SYSTEMS
13	21	ACM COMM	DIJKSTRA E W	65:09	SOLUTION IN CON CURRENT PROG CONTROL
14	22	ACM COMM	DIJKSTRA E W	68:05	STRUCTURE THE MULTIPROGRAMMING SYSTEM
15	23	ACM COMM	GRAHAM R M	68:05	PROTECTION INFORMATION PROCESSING
16	8	ACM COMM	HABERMAN A N	69:07	PREVENTION OF SYSTEM DEADLOCK
17	7	IBM SYSTEMS	HAVENDER J W	68:07	AVOIDING DEADLOCK MULTITASKING SYSTEM
18	15	COMP SURVEYS	HOFFMAN L J	69:06	COMPUTERS AND PRIVACY
19	5	CORNELL TMES	HOLT R C	71:06	DEADLOCK IN COMPUTER SYSTEMS
20	6	ACM COMM	IRONS E T	70:01	EXPERIENCE WITH EXTENSIBLE LANGUAGE
21	16	I-E TRANS	KILBURN:EDWARDS	62:04	ONE LEVEL STORAGE SYSTEM
22	18	AFIPS FJCC	LAMPSON B W	69:09	DYNAMIC PROTECTION STRUCTURES
23	19	PRINCETON	LAMPSON B W	71:03	PROTECTION
24	1	COMP SURVEYS	MACDOUGALL M H	70:09	COMPUTER SYSTEM SIMULATION
25	3	COMP SURVEYS	MCKINNEY J M	69:06	SURVEY ANALYTICAL TIME SHARING MODELS
26	4	ACM COMM	NIELSEN N R	70:08	ALLOCATION COMPUTER RESOURCES
27	24	OS SYMP	POOLE P:WAITE W	69:10	MACHINE INDEPENDENT SOFTWARE
28	25	ACM COMM	RANDELL:KUEHNER	68:05	DYNAMIC STORAGE ALLOCATION SYSTEMS
29	17	AFIPS SJCC	KLEINROCK L	70:05	CONTINUUM TIME SHARING SCHEDULING
30	27	COMP SURVEYS	ROSIN R F	69:03	SUPERVISORY AND MONITOR SYSTEMS
31	26	COMP SURVEYS	ROSIN S	69:03	ELECTRONIC COMPUTERS:HISTORICAL SURVEY
32	28	ACM COMM	SUTHERLAND I E	68:05	FUTURES MARKET IN COMPUTER TIME
33	29	COMP REVIEWS	TRIMBLE G JR	68:05	TIME SHARING BIBLIOGRAPHY
34	30	ACM COMM	WAITE W M	70:07	MOBILE PROGRAMMING SYSTEM:STAGE 2
35	35	ACM COMM	WILKES M V	68:01	COMPUTERS THEN AND NOW

(b) Sort by Author (ascending)

COUNT	POSITION		XXXXXXXXXXXXXXXXXXXXXXXXXXXX	SORT	
35	35	ACM COMM	WILKES M V	68:01	COMPUTERS THEN AND NOW
34	30	ACM COMM	WAITE W M	70:07	MOBILE PROGRAMMING SYSTEM:STAGE 2
33	29	COMP REVIEWS	TRIMBLE G JR	68:05	TIME SHARING BIBLIOGRAPHY
32	28	ACM COMM	SUTHERLAND I E	68:05	FUTURES MARKET IN COMPUTER TIME
31	26	COMP SURVEYS	ROSIN S	69:03	ELECTRONIC COMPUTERS:HISTORICAL SURVEY
30	27	COMP SURVEYS	ROSIN R F	69:03	SUPERVISORY AND MONITOR SYSTEMS
29	17	AFIPS SJCC	KLEINROCK L	70:05	CONTINUUM TIME SHARING SCHEDULING
28	25	ACM COMM	RANDELL:KUEHNER	68:05	DYNAMIC STORAGE ALLOCATION SYSTEMS
27	24	OS SYMP	POOLE P:WAITE W	69:10	MACHINE INDEPENDENT SOFTWARE
26	4	ACM COMM	NIELSEN N R	70:08	ALLOCATION COMPUTER RESOURCES
25	3	COMP SURVEYS	MCKINNEY J M	69:06	SURVEY ANALYTICAL TIME SHARING MODELS
24	1	COMP SURVEYS	MACDOUGALL M H	70:09	COMPUTER SYSTEM SIMULATION
23	19	PRINCETON	LAMPSON B W	71:03	PROTECTION
22	18	AFIPS FJCC	LAMPSON B W	69:09	DYNAMIC PROTECTION STRUCTURES
21	16	I-E TRANS	KILBURN:EDWARDS	62:04	ONE LEVEL STORAGE SYSTEM
20	6	ACM COMM	IRONS E T	70:01	EXPERIENCE WITH EXTENSIBLE LANGUAGE
19	5	CORNELL TMES	HOLT R C	71:06	DEADLOCK IN COMPUTER SYSTEMS
18	15	COMP SURVEYS	HOFFMAN L J	69:06	COMPUTERS AND PRIVACY
17	7	IBM SYSTEMS	HAVENDER J W	68:07	AVOIDING DEADLOCK MULTITASKING SYSTEM
16	8	ACM COMM	HABERMAN A N	69:07	PREVENTION OF SYSTEM DEADLOCK
15	23	ACM COMM	GRAHAM R M	68:05	PROTECTION INFORMATION PROCESSING
14	22	ACM COMM	DIJKSTRA E W	68:05	STRUCTURE THE MULTIPROGRAMMING SYSTEM
13	21	ACM COMM	DIJKSTRA E W	65:09	SOLUTION IN CON CURRENT PROG CONTROL
12	14	MIT MEMO	DENNIS J H	69:05	FUTURE TRENDS IN TIME SHARING SYSTEMS
11	11	ACM COMM	DENNIS J B	68:05	POSITION PAPER COMPUTING COMMUNICATNS
10	10	COMP SURVEYS	DENNING P J	70:09	VIRTUAL MEMORY
9	9	AFIPS FJCC	DENNING P J	68:09	THRASHING:ITS CAUSES AND PREVENTION
8	34	ACM COMM	DENNING P J	68:05	WORKING SET MODEL PROGRAM BEHAVIOR
7	2	ACM COMM	DALEY R:DENNIS J	68:05	VIRTUAL MEMORY SHARING IN MULTICS
6	33	AFIPS FJCC	CWITCHLOW A J	63:09	GENERALIZED MULTIPROGRAMMING SYSTEMS
5	32	IEEE INTNATL	CREECH R A	70:06	IMPLEMENTATION OF OPERATING SYSTEMS
4	31	DATAMATION	CORMATO F J	69:05	PL/I AS TOOL FOR SYSTEM PROGRAMMING
3	13	COMP SURVEYS	COFFMAN:ELPHICK	71:06	SYSTEM DEADLOCKS
2	12	ACM COMM	BERNSTEIN:SHARPE	71:02	POLICY DRIVEN SCHEDULER FOR TS
1	20	IEEE COMPUTER	AMATE J:DUNNER M	69:11	OPTIMIZING PERFORMANCE DRUM STORAGE

(c) Sort by Author (descending)

Figure 62. Example of Sort Routine Output

SORT

COUNT	POSITION	XXXXXXXXXXXXXXXXXXXXXXXXXX			
1	12	ACM COMM	BERNSTEIN:SHARPE	71:02	POLICY DRIVEN SCHEDULER FOR TSS
2	2	ACM COMM	DALEY R:DENNIS J	68:05	VERTUAL MEMORY SHARING IN MULTICS
3	34	ACM COMM	DENNING P J	68:05	WORKING SET MODEL PROGRAM BEHAVIOR
4	11	ACM COMM	DENNIS J B	68:05	POSITION PAPER COMPUTING COMMUNICATNS
5	21	ACM COMM	DIJKSTRA E W	65:09	SOLUTION IN CON CURRENT PROG CONTROL
6	22	ACM COMM	DIJKSTRA E W	68:05	STRUCTURE THE MULTIPROGRAMMING SYSTEM
7	23	ACM COMM	GRAHAM R M	68:05	PROTECTION INFORMATION PROCESSING
8	8	ACM COMM	HABERMAN A N	69:07	PREVENTION OF SYSTEM DEADLOCK
9	6	ACM COMM	IRONS E T	70:01	EXPERIENCE WITH EXTENSIBLE LANGUAGE
10	4	ACM COMM	NIELSEN N R	70:08	ALLOCATION COMPUTER RESOURCES
11	25	ACM COMM	RANDELL:KUEHNER	68:05	DYNAMIC STORAGE ALLOCATION SYSTEMS
12	26	ACM COMM	SUTHERLAND I E	68:06	FUTURES MARKET IN COMPUTER TIME
13	30	ACM COMM	WAITE W M	70:07	MOBILE PROGRAMMING SYSTEM:STAGE 2
14	35	ACM COMM	WILKES M V	68:01	COMPUTERS THEN AND NOW
15	33	AFIPS FJCC	CRITCHLOW A J	63:09	GENERALIZED MULTIPROGRAMMING SYSTEMS
16	9	AFIPS FJCC	DENNING P J	68:09	THRASHING:ITS CAUSES AND PREVENTION
17	18	AFIPS FJCC	LAMPSON B W	69:09	DYNAMIC PROTECTION STRUCTURES
18	17	AFIPS SJCC	PLEINROCK L	70:05	CONTINUUM TIME SHARING SCHEDULING
19	29	COMP REVIEWS	TRIMBLE G JR	68:05	TIME SHARING BIBLIOGRAPHY
20	13	COMP SURVEYS	COFFMAN:ELPHICK	71:06	SYSTEM DEADLOCKS
21	10	COMP SURVEYS	DENNING P J	70:09	VIRTUAL MEMORY
22	15	COMP SURVEYS	HOFFMAN L J	69:06	COMPUTERS AND PRIVACY
23	1	COMP SURVEYS	MACDOUGALL M H	70:09	COMPUTER SYSTEM SIMULATION
24	3	COMP SURVEYS	MCKINNEY J M	69:06	SURVEY ANALYTICAL TIME SHRING MODELS
25	27	COMP SURVEYS	ROSIN R F	69:03	SUPERVISORY AND MONITOR SYSTEMS
26	28	COMP SURVEYS	ROSIN S	69:03	ELECTRONIC COMPUTERS:HISTORICAL SURVEY
27	5	CORNELL THES	HOLT R C	71:06	DEADLOCK IN COMPUTER SYSTEMS
28	31	DATAMATION	CORBATO F J	69:05	PL/1 AS TOOL FOR SYSTEM PROGRAMMING
29	7	IBM SYSTEMS	HAVENDER J W	68:07	AVOIDING DEADLOCK MULTITASKING SYSTEM
30	20	IEEE COMPUTER	SMITH:JOURNER H	69:11	OPTIMIZING PERFORMANCE DRUM STORAGE
31	32	IEEE INTNATL	CREECH R A	70:06	IMPLEMENTATION OF OPERATING SYSTEMS
32	16	IRE TRANS	KILBURN:EDWARDS	62:04	ONE LEVEL STORAGE SYSTEM
33	14	MIT MEMO	DENNIS J B	69:06	FUTURE TRENDS IN TIME SHARING SYSTEMS
34	24	OS SYMP	POOLE P:WAITE W	69:10	MACHINE INDEPENDENT SOFTWARE
35	19	PRINCETON	LAMPSON B W	71:03	PROTECTION

(d) Sort by Journal (ascending)

Figure 62. Examples of Sort Routine Output (cont'd)

MATRIX A * MATRIX B = PRODUCT

MATRIX A

6	-12	-13	-7	-15	-3	
-15	6	13	-8	-10	-6	
11	4	-11	5	-4	-4	
-1	-11	2	10	2	13	
1	-11	14	-3	-11	1	

MATRIX B

11	0	22	-23	14	10		-9
18	-11	-10	-17	3	-6		2
7	-21	-7	21	20	16		-22
-3	-5	9	19	13	-8	6	-15
-14	-16	17	-24	-20	1	5	-12
-19	8	15	-1	23	-18	-13	12

PRODUCT

47	656	-20	23	-72	19	20	457
312	-187	-813	610	26	184	-130	29
233	194	196	-357	-1	-62	112	76
-500	101	393	381	382	-224	-63	-75
55	26	-165	664	465	295	-66	-150

Figure 63. Examples of Matrix Print Routine Output

MATRIX A * MATRIX B = PRODUCT

MATRIX A

-8	-1	2	9	10	1	12	1	5	-6	-7	5	11	0	9	14
-13	6	8	-3	-4	-13	10	3	5	-5	12	-8	9	9	12	10
11	-3	8	-6	13	7	-9	-5	7	-7	2	2	-6	-14	-6	4
8	6	9	14	8	-5	0	10	14	9	-5	14	-11	1	-13	-12
1	-5	8	-12	12	-11	4	0	1	-1	-4	-6	-6	2	8	-10
14	-9	-7	-12	10	3	7	-6	-14	2	10	13	10	3	9	13
-6	2	-2	-11	0	-1	2	2	-10	14	-1	-5	-1	-12	13	-1
-10	-12	7	-2	4	-6	13	-8	-6	7	1	3	-8	12	-6	13
-11	13	-10	-14	8	6	0	10	7	6	-14	6	-8	4	-7	-4
7	-13	-5	0	2	-2	2	13	-5	-6	-9	8	-10	9	11	1
6	0	-7	6	3	-3	-6	3	8	3	12	-14	12	1	-6	-10
-10	12	10	-12	10	-11	-13	-7	-14	-13	-14	5	-9	13	1	7
-2	-12	-11	12	-6	-13	14	-3	2	11	6	2	12	-14	-3	-3
2	0	-10	7	-7	13	10	5	10	11	1	11	8	-9	2	-8
-14	-10	3	9	-10	2	13	-2	-1	-12	-6	-5	7	-4	-13	11
10	-7	-6	0	4	-6	-2	5	-2	11	-7	1	-6	5	9	8

MATRIX B

6	0	-9	-8	12	-13	-8	4	-4	13	5	-8	-8	14	-5	-2
5	0	5	-10	8	-4	-6	7	11	4	-8	2	-14	13	12	-12
5	-10	-8	-5	9	8	10	6	3	8	11	2	-3	-8	-6	11
-13	6	-3	-1	-8	13	3	13	-10	6	9	10	11	11	9	3
-2	5	-12	-7	13	-5	7	-5	11	0	-2	1	-2	1	0	11
2	10	-3	-5	-2	1	-11	-6	-11	-1	7	9	-7	12	12	5
-13	-7	10	7	-5	-1	-5	12	14	-4	-2	7	-4	-1	-5	-7
-11	-8	2	14	-8	10	-1	5	2	1	-2	10	-2	14	-14	-3
-2	-2	-8	12	-7	5	-2	-8	9	-13	-3	-10	-9	-3	13	-10
7	-4	-2	-7	8	9	-3	10	-2	14	-14	1	-4	10	-3	0
0	-3	-12	5	13	5	-6	7	10	13	7	8	12	-2	14	13
2	4	-4	7	-4	-14	9	-1	-14	-5	-2	-5	12	-3	14	11
8	14	-1	1	-5	9	-1	9	-5	-8	6	-13	8	-6	7	11
3	-10	12	2	8	-9	7	5	-2	8	11	8	12	-7	-10	4
0	-12	11	12	-4	0	11	-14	12	-9	12	5	-10	-6	10	-14
6	-12	12	11	-10	-1	-4	-9	-13	-2	5	-1	-9	14	-14	7

PRODUCT

-243	-65	315	453	-523	141	242	-124	-6	-548	286	19	-79	0	61	112
-72	-628	494	592	-120	339	288	102	655	-190	341	180	9	-419	-97	-100
196	218	-691	-256	283	-222	-105	-437	-60	-8	9	-321	-344	134	156	308
-265	47	-607	-193	235	15	157	475	-1	345	-356	31	86	319	136	16
-24	-320	-69	-46	359	-62	370	-191	701	-57	-57	-39	-196	-508	-339	-209
330	-35	67	131	266	-606	-19	-168	-222	93	280	-158	149	26	-77	561
117	-177	212	-42	46	169	-15	-149	333	-22	-332	78	-411	62	-82	-308
-69	-472	316	140	55	-108	239	101	-100	174	43	234	300	-320	-640	423
-11	44	199	-38	-61	-274	-71	-239	166	-303	-712	9	-389	202	-72	-386
-290	-359	319	435	-243	-371	329	-301	-172	-119	232	226	83	43	-502	-154
-8	339	-441	-156	264	394	-273	357	261	238	-7	-114	222	50	239	52
352	-295	421	-288	302	-644	614	-441	-13	-72	51	-26	-46	-421	-493	118
-327	270	-112	248	-399	463	-138	500	-11	-115	-283	-203	421	-100	197	20
-190	378	-152	202	-382	234	-319	239	-148	-216	-228	-43	-18	334	692	-188
-361	129	323	282	-748	397	-119	170	-390	-401	172	41	264	-176	-326	220
28	-359	219	149	5	-191	131	-179	-143	148	-48	11	-180	305	-487	-165

Figure 63. Examples of Matrix Print Routine Output (cont'd)

APPENDIX I

HISTORICAL REVIEW OF MICROPROGRAMMING

Digital computing systems have traditionally been described as being composed of the five basic units: input, output, memory, arithmetic/logic, and control (Figure 64). Machine instructions and data are communicated among these units as indicated by the heavy lines in the figure are generally well known and understood. The control signals (as indicated by light lines in the figure), are generally less well known and understood except by the system designer. These control signals generated in the control unit determine the information flow and timing of the system.

Microprogramming is a term associated with the orderly and systematic approach to the design of the control unit. The functions of the control unit include:

1. Fetching the next machine instruction to be executed from memory
2. Decoding the machine instruction and providing each microstep control
3. Controlling the gating of data paths to perform the specified operation
4. Changing the machine state to allow fetching of the next instruction.

The conventional control unit is designed using flip-flops (e.g., registers and counters) and gating in a relatively irregular ad hoc manner. By contrast the control unit of a microprogrammable computer is implemented using well structured memory elements, thus providing a means for well organized and flexible control.

Microprogramming is therefore a technique for implementing the control function of a digital computing system as sequences of control signals that are organized on a word basis and stored in a memory unit.

It should be noted that if this memory is alterable, then microprogramming allows the modification of the system architecture as observed at the machine language level. Thus, the same hardware may be made to appear as a variety of system structures; thereby achieving optimum processing capability for each task to be performed. The ability to alter the microprogram memory is called dynamic microprogramming as compared to static microprogramming which uses read only memories.

As can be seen in the following brief historical review, the concept of microprogramming was not widely accepted except academically during the 1950's. The primary reason for this was its high cost of implementation, especially the cost of control memories. From the mid-1960's to the present there has been a definite trend toward microprogrammable processors and more recently to dynamic microprogramming. This effort has been inspired by rapid advances in technology, especially control memories.

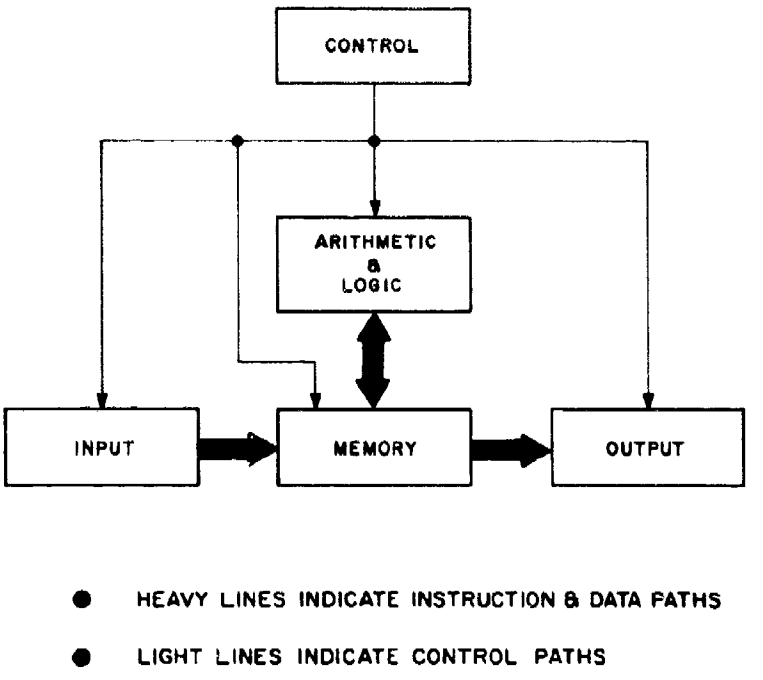


Figure 64. Traditional Digital Computing System Block Diagram

BRIEF HISTORICAL REVIEW OF MICROPROGRAMMING

1951	Wilkes[1] objective was "to provide a systematic approach and an orderly approach to designing the control section of any computing system." He likened the execution of the individual steps within a machine instruction to the execution of the individual instructions in a program; hence the term microprogramming. This view is hardware design oriented.
	Lincoln Lab (see Van der Poel[2]) with different emphasis used the term microprogramming to describe a system in which the individual bits in an instruction directly control certain gates in the processor. The objective here was to provide the programmer with a larger instruction repertoire. This view is software design oriented.
1956/7	Glantz[3] and Mercer[4] pointed out that through microprogram modifications the processor instruction set may be varied.
1958-1960	Blankenbaker[5], Dinneen[6], and Kampe[7] described simple computers based on Wilkes model.
1961-1964	Great international interest was shown from U.S., U.K., Italy, Japan, Russia, Australia and France.
Feb. 1964	In Datamation[8-12] five articles appeared on microprogramming with emphasis on how it might extend the computing capacity of small machines.
1964	IBM System 360 (Stevens[13]) demonstrated that through microprogramming, computers of different power with compatible instruction sets could be provided (used read only storage).
1965	Melbourne and Pugmire[14] described microprogramming support for compiling and interpreting higher level programming languages.

1965	McGee and Petersen[15] pointed out the advantage of using an elementary microprogrammed computer as a peripheral controller; i.e., as an interface between computers and peripheral devices.
1965-1966	Green[16], and Tucker[17] described emulation of one machine on another through microprogramming.
1967	Opler[18] coined the term "firmware" for microprograms designed to support software and suggests the increased usage of microprogramming and describes its advantages.
1967	Hawryszkiewycz[19] discussed microprogram support through special instructions for problem oriented languages.
1967	Rose[20] described a microprogrammed graphical interface computer.
1968	Lawson[21] discussed program language oriented instruction streams.
1969	Wilkes[22] and Rosin[23] provided surveys of the microprogramming advances.
	There were also announcements of many new microprogrammed computers (e.g., Standard Computer - Rakoczi[24]).
1970	Husson[25] provided the first textbook on microprogramming.
1971	Tucker and Flynn[26] pointed out advantages of adapting the machine to the task through microprogramming.
July 1971	The IEEE Transactions on Computers offered a special issue on microprogramming.
July 1972	Clapp[27] and Jones, et. al.[28] provide annotated microprogramming bibliographies.

1. Wilkes, M. V. "The Best Way to design an Automatic Calculation Machine" Manchester University Computer Inaugural Conference Proceeding (1951), p. 16.

2. Van Der Poel, W. L. "Micro-Programming and Trickology" John Wiley and Sons, Inc. (1962), Digital Information Processors.

3. Glantz, H. T. "A Note on Microprogramming" Journal ACM 3, Vol. No. 2, (1956), p. 77.

4. Mercer, R. J. "Micro-Programming" Journal ACM 4, Vol. No. 2 (1957), p. 157.

5. Blankenbaker, J. V. "Logically Microprogrammed Computers" IRE Prof. Group on Elec. Com. (December 1958), Vol. EC-7, No. 2, pp. 103-109.

6. Dineen, G. P., Lebow, I. L., et al. "The Logical Design of CG24" Proc. E. J. C. C. (December 1958), pp. 91-94.

7. Kampe, T. W. "The Design of a General-Purpose Microprogram-Controlled Computer with Elementary Structure" IRE Trans. (June 1960), Vol. EC-9, No. 2, pp. 208-213.

8. Beck, L., Keeler, F. "The C-8401 Data Processor" (February 1964). Datamation, pp. 33-35.

9. Boutwell, Jr., O. "The PB 440 Computer" (February 1964), Datamation, pp. 30-32.

10. Amdahl, L. D. "Microprogramming and Stored Logic" (February 1964), Datamation, pp. 24-26.

11. Hill, R. H. "Stored Logic Programming and Applications" (February 1964), Datamation, pp. 36-39.

12. McGee, W. C. "The TRW-133 Computer" (February 1964), Datamation, pp. 27-29.

13. Stevens, W. Y. "The Structure of SYSTEM/360 Part II - System Implementation" IBM Systems Journal, Vol. 3, No. 2 (1964) pp. 136-143.

14. Melbourne, A. J., Pugmire, J. M., et al. "A Small Computer for the Direct Processing of Fortran Statements" Computer Journ. (England) (April 1965), Vol. 8, No. 1, pp. 24-27.

15. McGee, W. C. and Peterson, H. E. "Microprogram Control for the Experimental Sciences" Proc. AFIPS (1965), FJCC Vol. 27, pp. 77-91.

16. Green, J. "Microprogramming Emulators and Programming Languages" Comm. of ACM (March 1966), Vol. 9, No. 3, pp. 230-232.

17. Tucker, S. G. "Emulation of Large Systems" Communications of the ACM (December 1965), Vol. 8, No. 12, pp. 753-761.

18. Opler, A. "Fourth-Generation Software, the Realignment" Datamation (January, 1967), Vol. 13, No. 1, pp. 22-24.

19. Hawryszkiewycz, I. T. "Microprogrammed Control in Problem-Oriented Languages" IEEE Transactions on Electronic Computers (October 1967), Vol. EC-16, No. 5, pp. 652-658.

20. Rose, G. A. "Intergraphic, a Microprogrammed Graphical-Interface Computer" IEEE Transactions (December 1967), Vol. EC-16, No. 6, pp. 776-784.

21. Lawson, H. W. "Programming Language-Oriented Instruction Streams" IEEE Transactions (1968), C-17, p. 476.

22. Wilkes, M. V. "The Growth of Interest in Microprogramming - A Literature Survey" Comp. Surveys, Vol. 1, No. 3 (September 1969), pp. 139-145.

23. Rosin, R. F. "Contemporary Concepts of Microprogramming and Emulation" Comp. Surveys, Vol. 1, No. 4 (December 1969), pp. 197-212.

24. Rakoczi, L. L. "The Computer-Within-a-Computer: A Fourth Generation Concept" Computer Group News, Vol. 2, No. 8, (March 1969), pp. 14-20.

25. Husson, S. "Microprogramming: Principles and Practices" Prentice Hall, Englewood Cliffs, N. J. (1970).

26. Tucker, A. B. and Flynn, M. J. "Dynamic Microprogramming: Processor Organization and Programming" CACM (April 1971), Vol. 14, No. 4, pp. 240-250.

27. Clapp, J. A. "Annotated Microprograming Bibliography" SIGMICRO Newsletter, Vol. 3, Issue 2, (July 1972), pp. 3-38.

28. Jones, L. H., Carvin, K. et al. "An Annotated Bibliography on Microprogramming" SIGMICRO Newsletter, Vol. 3, Issue 2, (July, 1972), pp. 39-55.

APPENDIX II

FINAL SUMMARY REPORT

BIPOLAR LSI

FOR

BURROUGHS INTERPRETER

MAY 1972

CONTRACT NO. 82329

PREPARED BY

TEXAS INSTRUMENTS INCORPORATED

P. O. BOX 1443

HOUSTON, TEXAS 77001

FOR

BURROUGHS CORPORATION

DEFENSE, SPACE & SPECIAL SYSTEMS GROUP

PAOLI, PENNSYLVANIA 19301

TABLE OF CONTENTS

cont'd ...

TABLE OF CONTENTS

(cont'd)

LARGE SCALE INTEGRATION

Via Discretionary Routed Arrays

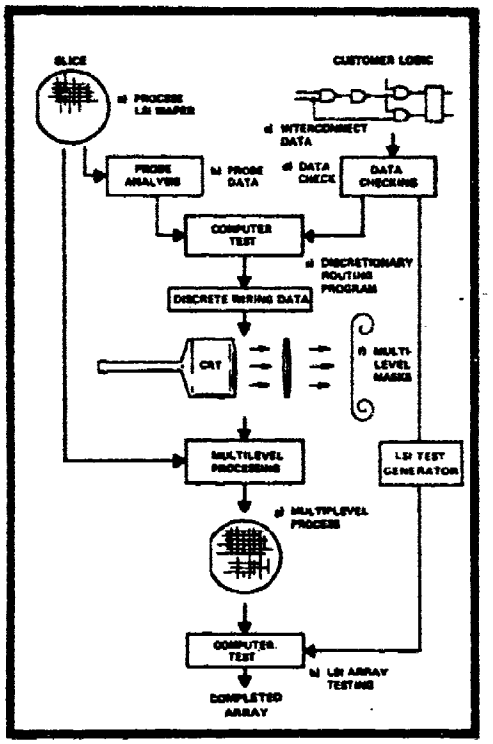


Figure 1. Multilevel process from wafer to array test, all computerized.

Texas Instruments is using monolithic discretionary routing technology to produce Large Scale Integrated (LSI) arrays. Large bipolar wafers are produced containing an intermix of the gates and flip-flops required to perform logic functions.

More than 16,000 separate components are diffused into a single 1 1/2-inch-diameter silicon slice. These components are then connected with first level metallization into a minimum of 1410 equivalent gates. (See Figure 1.) The slice is then probed to determine the individual characteristics of each device on the slice.

Customer logic requirements are fed into computer-controlled equipment, which has been developed to generate unique interconnection masks for each wafer at low cost.

Custom interconnections are then produced using probe test data and a computer to develop the discretionary routing masks. Using these automated techniques, custom arrays can be developed to fit most logic specifications. Multilevel metal interconnect technology now makes possible the production of very complex arrays in a short time.

CUSTOM LSI ARRAYS

Custom LSI Arrays are produced by discretionarily interconnecting various circuits or cell types on the face of an LSI wafer, similar to the interconnection of individual integrated circuits on a PC board. These are TTL logic circuit types and are similar to TI standard series SN5400 integrated circuits. The same general logic rules (loading, fan-in, fan-out, logic states, speeds, etc.) that apply to series SN5400, apply to the LSI circuits. Therefore, to design a system with LSI, or to reimplement an existing one, is a relatively easy, straightforward process.

LSI INTERFACE

There are three basic interface methods that can be achieved with the LSI technology:

1) The first method is to implement a functional bipolar logic requirement with the standard wafers currently in assignment inventory, shown on page . These types are currently in production and stocked, waiting for assignment to a logic requirement. The addition of multilevel metallization converts these slices into functional arrays.

   Partitioning the arrays for the number of circuits and types available on the wafer and limiting the number of input-outputs, not to exceed 126, is all that is required. Presently, the time from logic diagram input to completed array is in the range of 30 to 90 days, depending on complexity.

2) The second interface method is implemented by creating a custom wafer using standard circuits from our circuits library. This often reduces the total number of arrays needed in a system, thus reducing the system cost. The highest single cost in the design of IC's is the set of diffusion masks used to create the individual circuits. This high cost has already been absorbed in the design of standard circuits. Stepping and repeating these standard circuits around on a wafer to form a custom distribution or quantity of given circuit types is a relatively low-cost operation. Thus, a custom wafer containing a unique distribution of circuits for a specific application provides the interface.

   TI is continuously expanding the present circuits library with new, more complex circuits. Most of these will be similar, if not identical, to the circuits presently available as standard Series 5400. Thus, implementing LSI arrays remains simple.

3) The third interface method with LSI is a total custom approach. A few thousand arrays of a single type may justify the expense of a custom circuit as well as that of a unique wafer. General-purpose logic arrays will provide 200- to 800-gate complexity while customized circuits and wafers can provide arrays of 500- to over 2000-gate complexity on a single monolithic substrate.

ARRAY TESTING

The final phase of creating an LSI array is the testing of interconnections and the verification that the array will perform in accordance with the logic diagram. Because testing an input logic array with all possible combinations of inputs that can occur is impractical, TI has developed a "single-fault modeling" approach. Testing for a single type of fault at each node within the logic network is both practical and effective. This approach assumes that a set of inputs can be defined that not only will exercise each circuit output but also will test for the output being stuck-at-one or stuck-at-zero.

The number of tests required for a 200- to 400-gate array is in the thousands. But this is a reasonable number to generate and test with computer programs and computer-controlled test equipment. The equipment is capable of applying 5,000 tests per second to a 156-pin LSI package.

This approach to tests does not require knowledge of the functional capability of a logic array. Therefore, a logic diagram can be provided, the multilevel interconnection accomplished, and the completed array tested without the operator knowing what the array does functionally. This gives the customer confidence that his circuit innovations are protected. In addition, it assures that this information is treated on a proprietary basis.

ARRAY PACKAGE

A general-purpose package has been developed for housing whole wafers of monolithic semiconductor components. The package serves as a suitable container, protects the wafer from handling and environments, provides for adequate heat transfer, and is capable of mounting and interconnection into customers' equipment. A 2 1/8-inch square, alumina-ceramic substrate with thick-film metallization leads is the package developed through extensive research. It provides 39 leads on 50-mil centers on all 4 sides of the package so that conventional solder or reflow solder techniques can be used.

Normally the wafer is mounted with a special high-temperature epoxy adhesive, providing typically a 3° C/W gradient between the LSI wafer and the ceramic header. The wafer is connected to the gold-plated lead frame with gold wires, using conventional thermocompression techniques. This results in a high-reliability all gold system. The standard package has an epoxy-sealed ceramic lid, but a hermetically sealed package with Kovar-type lid can be provided.

LOGIC DRAWINGS

1. LOGIC UNIT 1
   REV. B - 12-14-70
   SK-0982-0109
   DRA-3013

2. LOGIC UNIT 2
   REV. F - 3-4-71
   SK-0982-0110
   DRA-3014

3. CONTROL UNIT 1
   REV. D - 4-16-71
   SK-0982-0113
   DRA-3015

4. CONTROL UNIT 2
   REV. B - 4-26-71
   SK-0982-0114
   DRA-3016

5. MEMORY CONTROL UNIT 1
   REV. D - 4-16-71
   SK-0982-0111
   DRA-3017

6. MEMORY CONTROL UNIT 2
   REV. C - 4-27-71
   SK-0982-0112
   DRA-3018

AA - 5/8/72

GENERAL CIRCUIT CHARACTERISTICS

absolute maximum ratings over operating case temperature range (unless otherwise noted)

Supply Voltage V_{CC} Short Duration (30 seconds) (see note 1) . . . . . 7 V
Input Voltage V_I (see notes 1 and 2) . . . . . 5.5 V
Operating Case Temperature Range . . . . . −55°C to 125°C
Storage Temperature Range . . . . . −65°C to 150°C

NOTES: 1. Voltages are with respect to network ground terminal.
2. Input signals must be zero or positive with respect to network ground terminal.

recommended operating conditions

	MIN	TYP	MAX	UNIT
Supply Voltage V_{CC}	4.5	5	5.5	V

electrical characteristics over operating temperature range (unless otherwise noted)

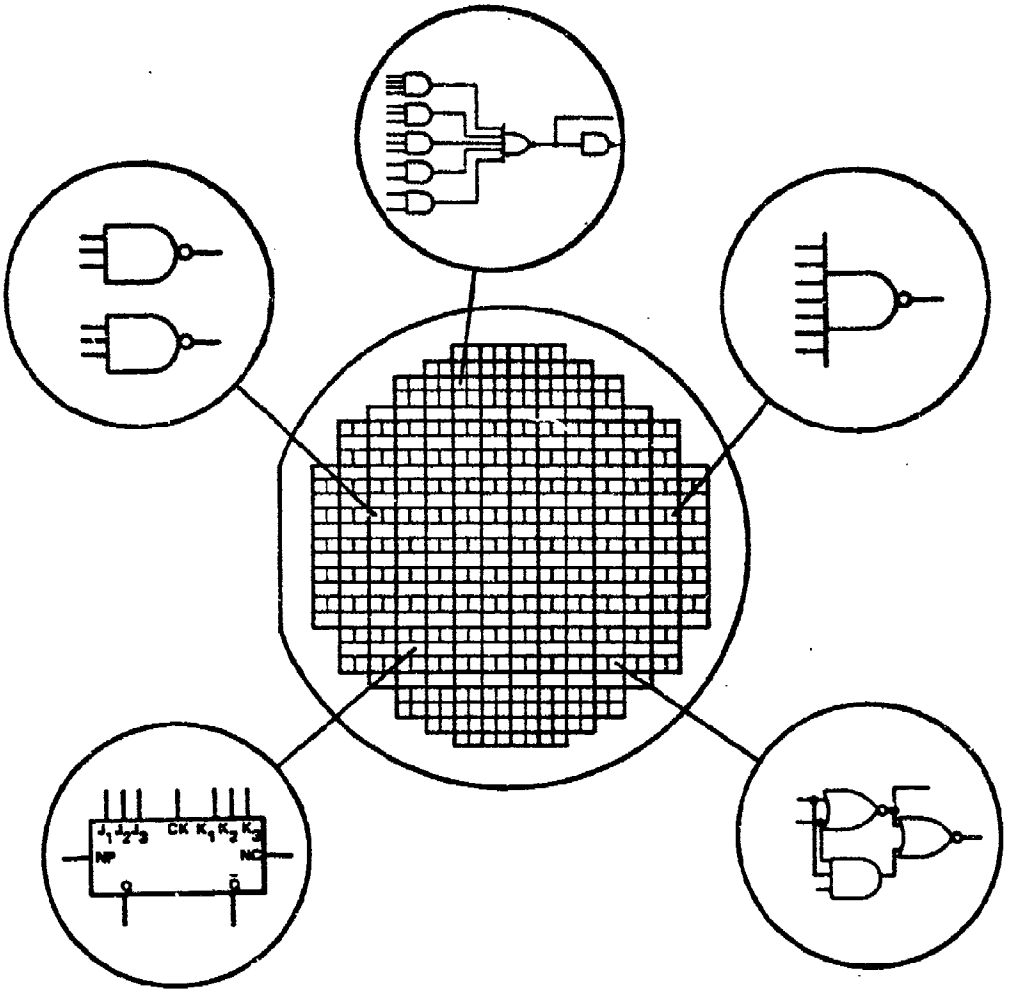
PARAMETER		TEST CONDITIONS		MIN	TYP*	MAX	UNIT
V_{IH}	High level input voltage	V_{CC} = 4.5 V		2			V
V_{IL}	Low level input voltage	V_{CC} = 4.5 V				0.8	V
V_{OH}	High level output voltage	V_{CC} = 4.5 V,	I_{load} = 400 μA	2.4	3.5		V
V_{OL}	Low level output voltage	V_{CC} = 4.5 V,	I_{sink} = 8 mA		0.22	0.4	V
I_{IH}	High level input current one normalized load	V_{CC} = 5.5 V,	V_{IH} = 2.4 V,			40	μA
		V_{CC} = 5.5 V,	V_{IH} = 5.5 V			1	mA
I_{IL}	Low level input current one normalized load	V_{CC} = 5.5 V,	V_{IL} = 0.4 V			−1.6	mA
I_{OS}**	Short-circuit output current (output in logic one state)	V_{CC} = 5.5 V		−18		−57	mA

* All typical values are at V_{CC} = 5 V, T_A = 25°C
** Not more than one output should be shorted at a time.

fan-out

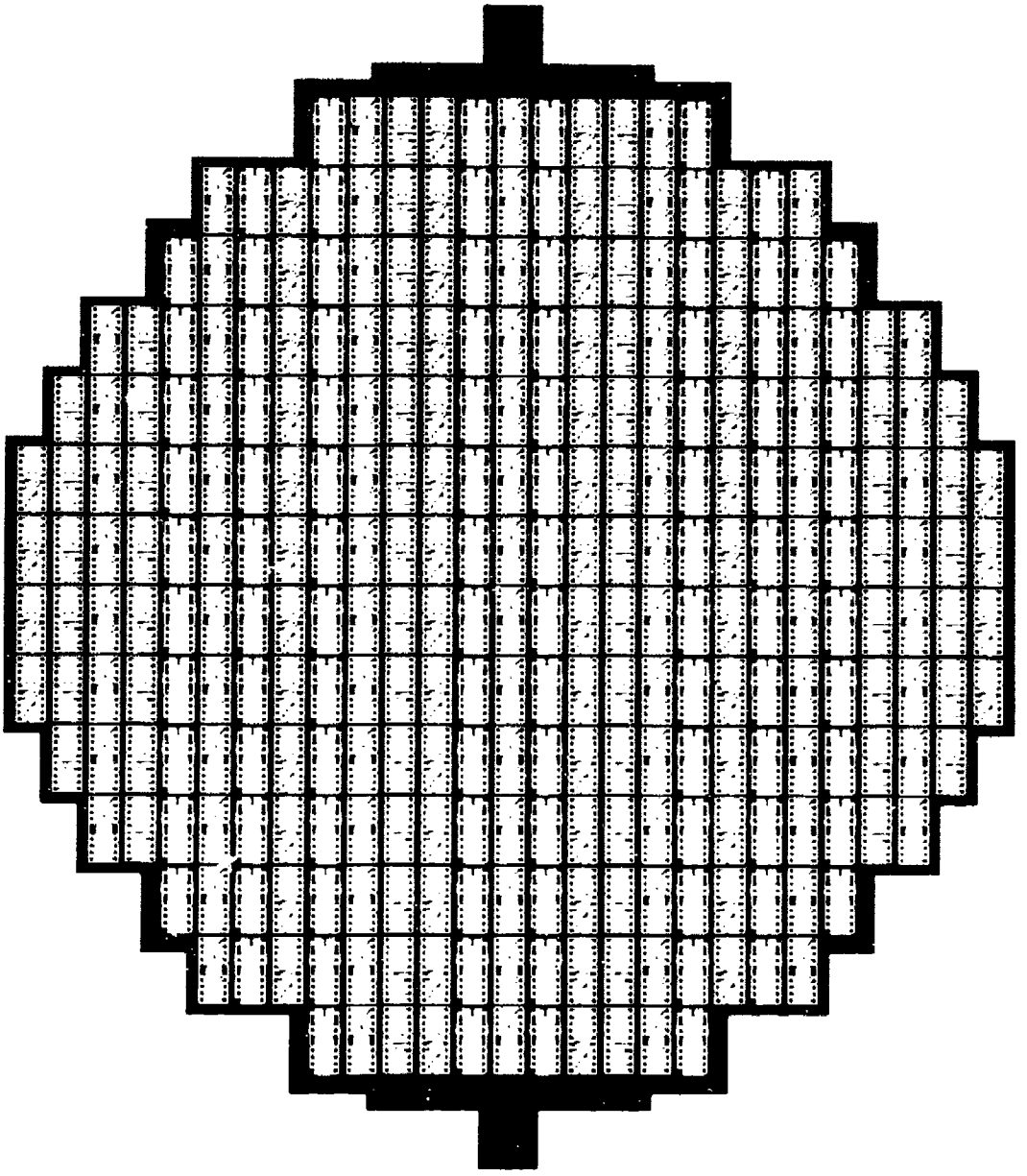
All LSI gates and flip-flops are rated for a normalized fan-out of 10. This fan-out should not include more than 5 external (outside of package) 0 state loads.

LOGIC SLICE – TYPE "N"

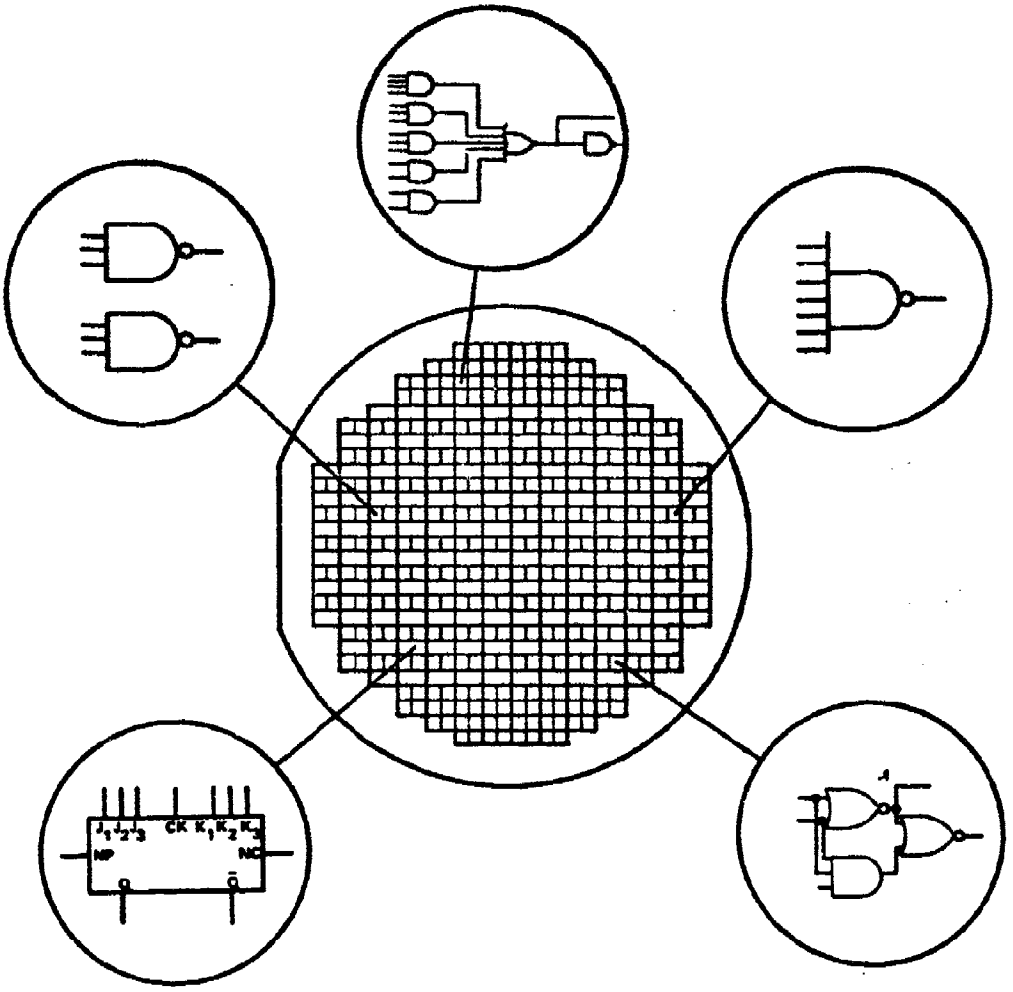


CIRCUIT TYPE	TOTAL NO.	RECOMMENDED* MAXIMUM USE
J-K FLIP-FLOP	100	30
AND-NOR-INVERT GATE	82	25
EXCLUSIVE OR GATE	60	18
3-INPUT GATE	232	70
7-INPUT GATE	56	17

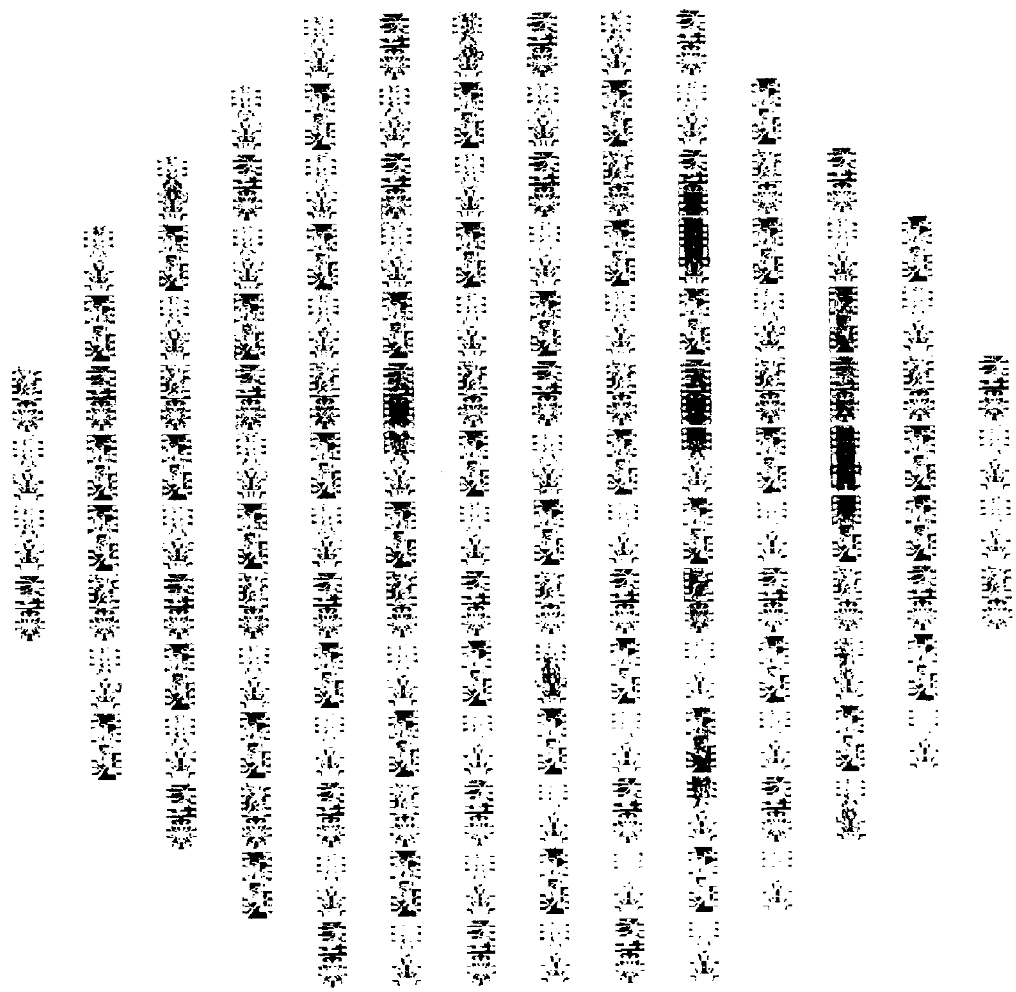
* RECOMMENDED DESIGN WITH UP TO 30% OF EACH SINGLE-CIRCUIT TYPE.



LOGIC SLICE – TYPE "S"



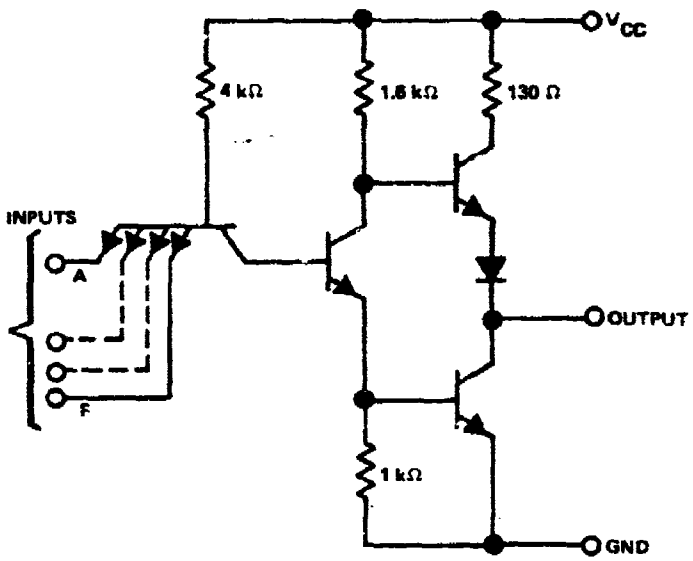
CIRCUIT TYPE	TOTAL NO.	RECOMMENDED MAXIMUM USE
J-K FLIP-FLOP	58	26
AND-NOR-INVERT GATE	46	21
EXCLUSIVE OR GATE	18	10
3-INPUT GATE	96	60
7-INPUT GATE	30	19



NAND GATE

LOGIC

SCHEMATIC



COMPONENT VALUES SHOWN ARE NOMINAL.

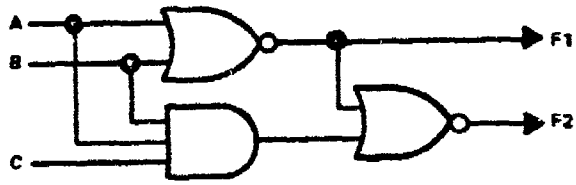
CHARACTERISTICS (V_{CC} = 5 V, T_A = 25°C, N = 10)

PARAMETER	MIN	TYP	MAX	UNIT
AV PROPAGATION DELAY		9	19	ns
POWER DISSIPATION		10		mW
FAN-IN (NORMALIZED)			1	–
FAN-OUT (NORMALIZED)	10			–

NOTE: FOR MORE GATE INFORMATION SEE SN5400 DATA SHEET.

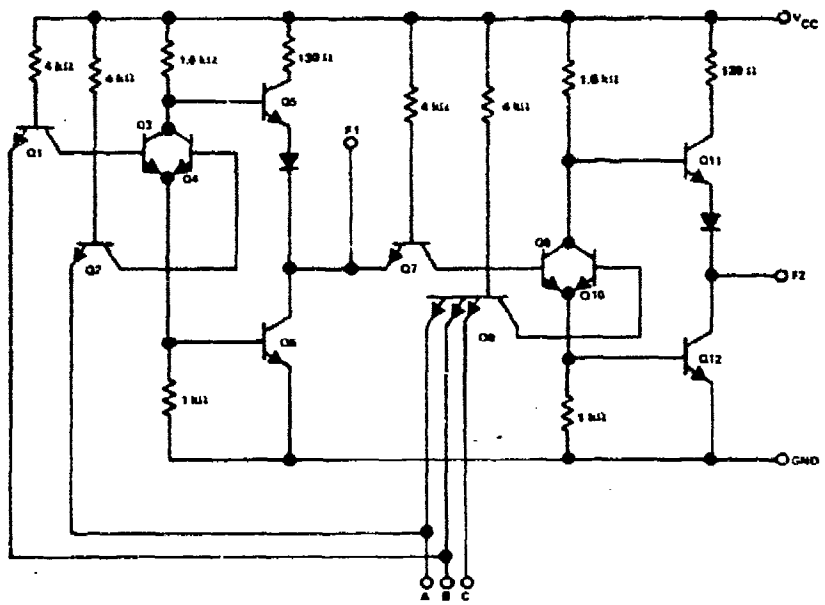
EXCLUSIVE OR GATE

LOGIC



INPUTS			OUTPUTS	
A	B	C	F1	F2
0	0	0	1	0
1	0	0	0	1
0	1	0	0	1
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	0	0

SCHEMATIC

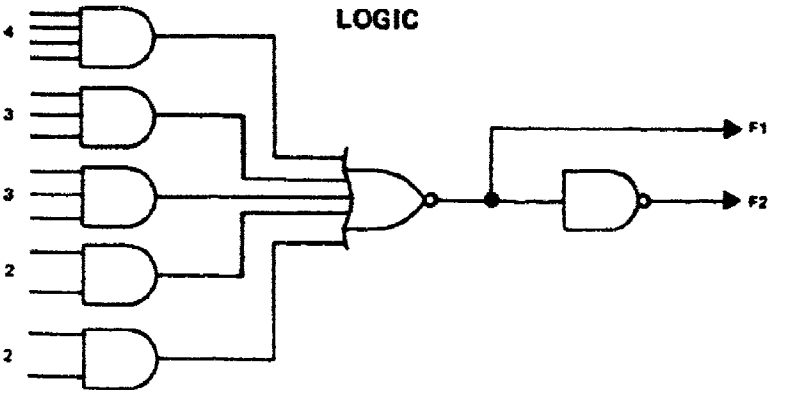


CHARACTERISTICS (V_{CC} = 5 V, T_A = 25°C, N = 10)

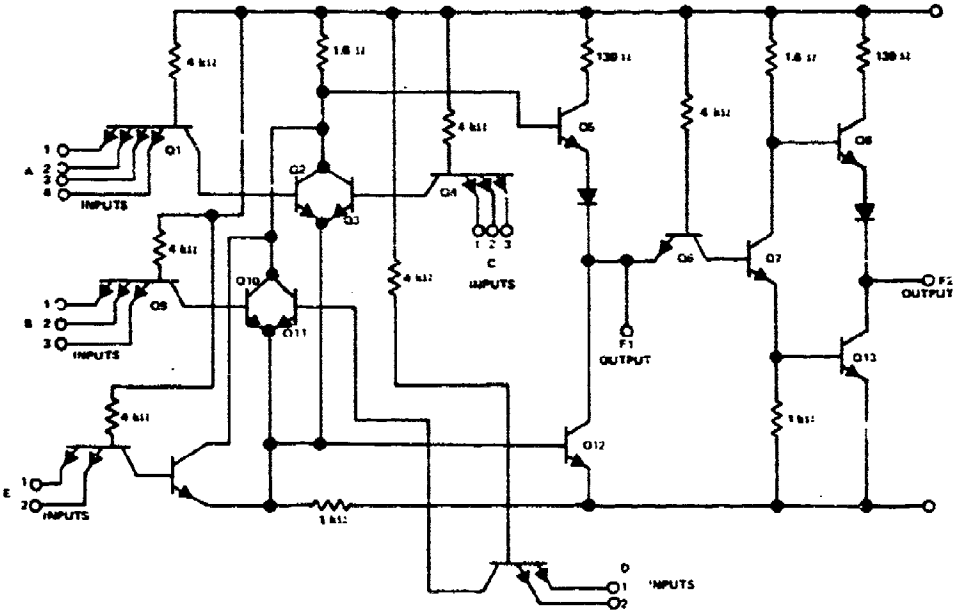
PARAMETER	MIN	TYP	MAX	UNIT
AV PROPAGATION DELAY				
F1		9	19	ns
F2		18	38	ns
POWER DISSIPATION		26		mW
FAN-IN (NORMALIZED)				
A & B			2	—
C			1	—
FAN-OUT (NORMALIZED)	10			—

AND-NOR-INVERT GATE

LOGIC



SCHEMATIC

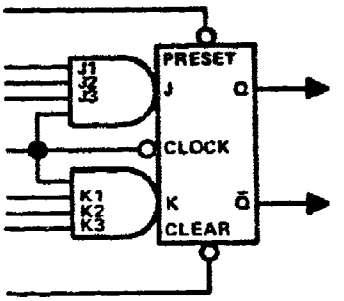


CHARACTERISTICS (V_{CC} = 5 V, T_A = 25°C, N = 10)

PARAMETER	MIN	TYP	MAX	UNIT
AV PROPAGATION DELAY				
F1		10	20	ns
F2		19	39	ns
POWER DISSIPATION		40		mW
FAN-IN (NORMALIZED)			1	–
FAN-OUT (NORMALIZED)	10			–

J-K MASTER-SLAVE FLIP-FLOP

LOGIC



TRUTH TABLE		
t_n		t_{n+1}
J	k	Q
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

POSITIVE LOGIC:

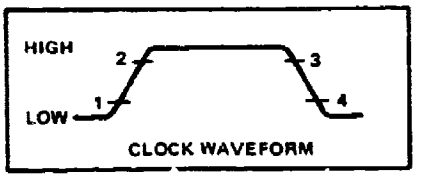
LOW INPUT TO PRESET SETS Q TO LOGICAL 1
LOW INPUT TO CLEAR SETS Q TO LOGICAL 0
PRESET AND CLEAR ARE INDEPENDENT OF CLOCK

NOTES: 1. $J = J1 \cdot J2 \cdot J3$
2. $K = K1 \cdot K2 \cdot K3$
3. t_n = BIT TIME BEFORE CLOCK PULSE.
4. t_{n+1} = BIT TIME AFTER CLOCK PULSE.

DESCRIPTION

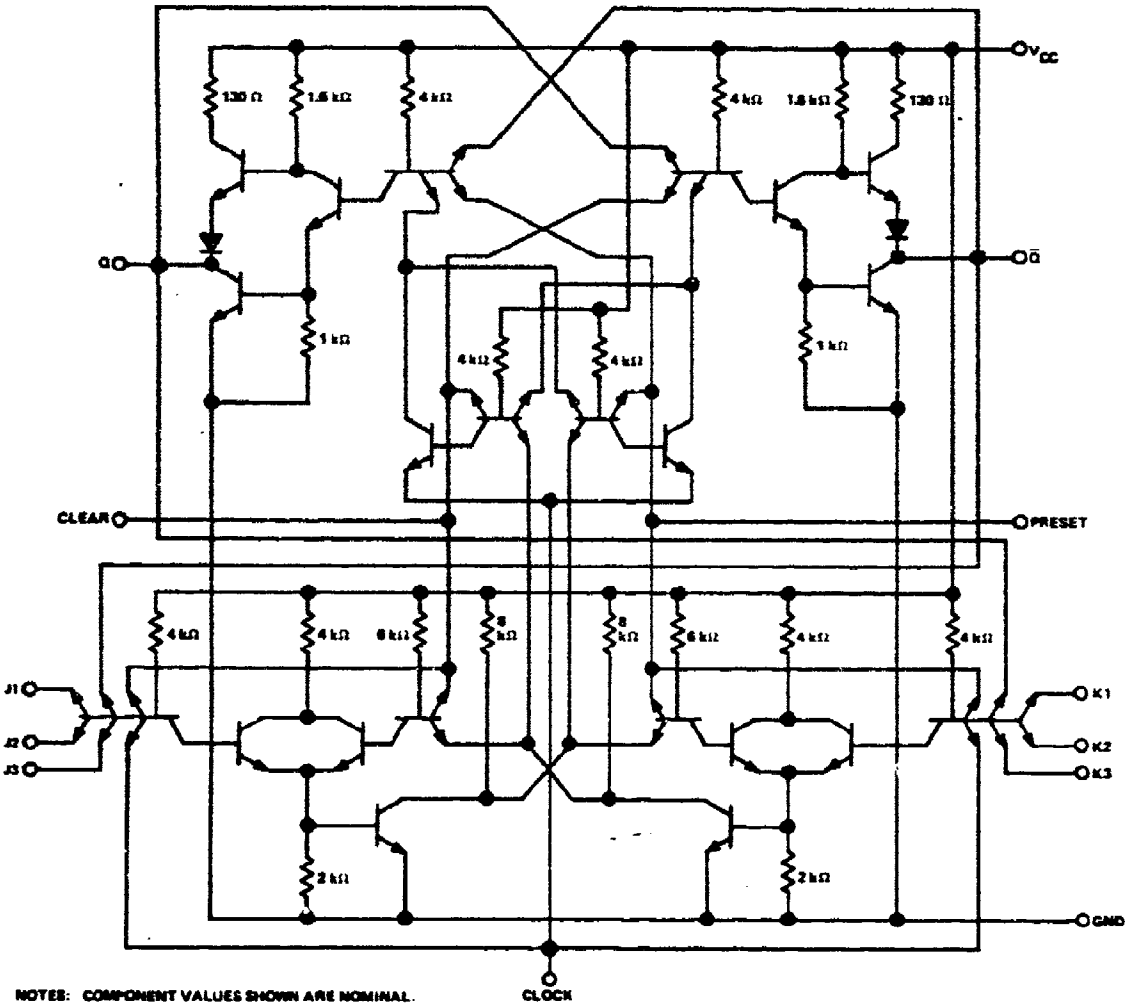
THESE J-K FLIP-FLOPS ARE BASED ON THE MASTER-SLAVE PRINCIPLE AND EACH HAS AND GATE INPUTS FOR ENTRY INTO THE MASTER SECTION WHICH ARE CONTROLLED BY THE CLOCK PULSE. THE CLOCK PULSE ALSO REGULATES THE STATE OF THE COUPLING TRANSISTORS WHICH CONNECT THE MASTER AND SLAVE SECTIONS. THE SEQUENCE OF OPERATION IS AS FOLLOWS:

1. ISOLATE SLAVE FROM MASTER
2. ENTER INFORMATION FROM AND GATE INPUTS TO MASTER
3. DISABLE AND GATE INPUTS
4. TRANSFER INFORMATION FROM MASTER TO SLAVE.



J-K MASTER-SLAVE FLIP-FLOP (CONTINUED)

SCHEMATIC



NOTES: COMPONENT VALUES SHOWN ARE NOMINAL.

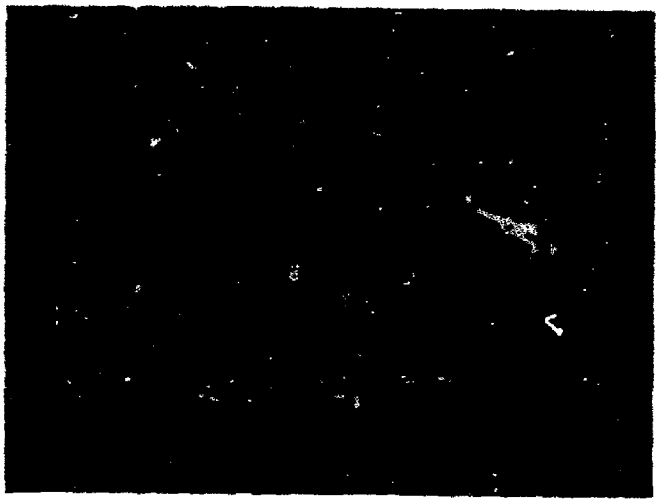
CHARACTERISTICS (V_{CC} = 5 V, T_A = 25°C, N = 10)

PARAMETER	MIN	TYP	MAX	UNIT
MAX CLOCK FREQUENCY	15	20		MHz
POWER DISSIPATION		40		mW
FAN-IN (NORMALIZED)				
J & K			1	–
PRESET, CLEAR & CLOCK			2	–
FAN-OUT	10			–

NOTE: FOR MORE FLIP-FLOP INFORMATION REFER TO SN5472 DATA SHEET.

DUAL 3-INPUT NAND GATE

7-INPUT NAND GATE



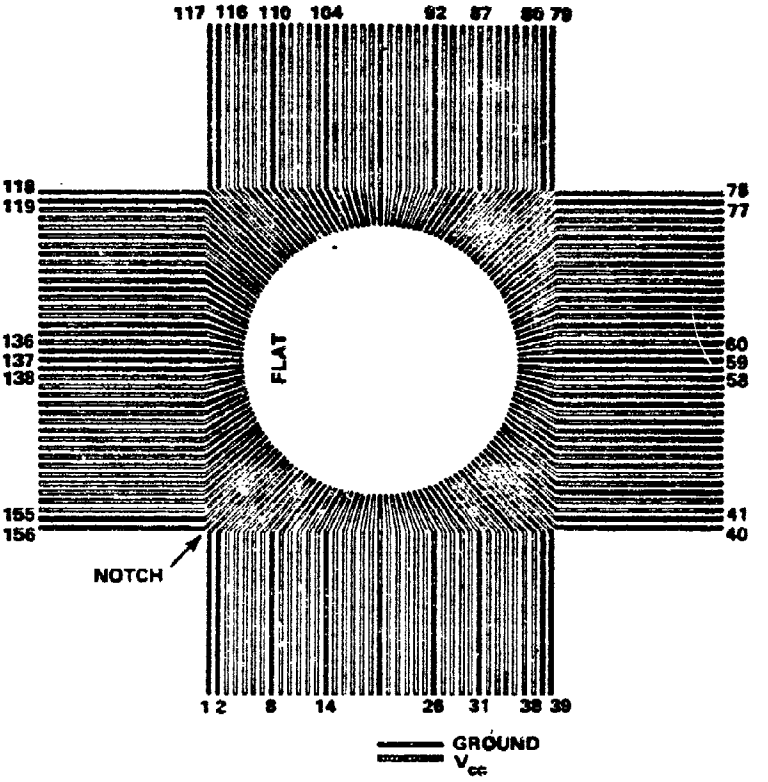
EXCLUSIVE OR GATE

AND - NOR - INVERT GATE

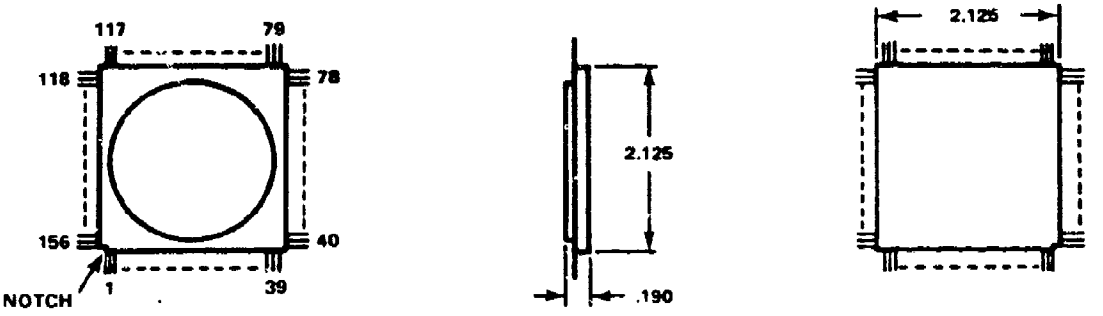
J - K MASTER-SLAVE FLIP-FLOP

PACKAGE DATA

PIN LAYOUT



PACKAGE DIAGRAM



TOP VIEW

BOTTOM VIEW

LEADS ARE GOLD PLATED F-15 ALLOY ON 50 MIL CENTERS

DIMENSIONS NOMINAL (IN INCHES)

WEIGHT – 22 GRAMS

DRA-3013 SUMMARY

Function	Total No.	Gate Complexity	Total Gate Complexity	Pins Used	Pins Available	Percent Used
FF	8	6	48	40	88	45
ANI	30	7	210	265	480	55
EXOR	18	3	54	64	90	71
3G	93	1	93	257	372	69
7G	15	1	15	82	120	68
TOTALS:	164		420	708	1150	61

POWER DISSIPATION - 3.14 WATTS

TOTAL PINS - 801, including 93 I/O PINS

INPUT CONNECTOR PINS - 67

OUTPUT CONNECTOR PINS - 26

AA - 5/8/72

DRA-3014 SUMMARY

Function	Total No.	Gate Complexity	Total Gate Complexity	Pins Used	Pins Available	Percent Used
FF	32	6	192	128	352	36
ANI	26	7	182	247	416	59
EXOR	16	3	48	64	80	80
3G	113	1	113	344	452	76
7G	17	1	17	103	136	75
TOTALS:	204		552	886	1436	61

POWER DISSIPATION - 4.10 WATTS

TOTAL PINS - 967, including 81 I/O PINS

INPUT CONNECTOR PINS - 47

OUTPUT CONNECTOR PINS - 34

AA - 5/8/72

DRA-3015* SUMMARY

Function	Total No.	Gate Complexity	Total Gate Complexity	Pins Used	Pins Available	Percent Used
FF	22	6	132	109	242	45
ANI	21	7	147	128	336	38
EXOR	9	3	27	30	45	66
3G	83	1	83	224	332	67
TOTALS:	135		389	491	955	51

POWER DISSIPATION - 2.82 WATTS

TOTAL PINS - 559, including 71 I/O PINS

INPUT CONNECTOR PINS - 36

OUTPUT CONNECTOR PINS - 35

*NOTE - 2-LEVEL METAL SYSTEM

AA - 5/8/72

DRA-3016* SUMMARY

Function	Total No.	Gate Complexity	Total Gate Complexity	Pins Used	Pins Available	Percent Used
FF	14	6	84	62	154	40
ANI	25	7	175	157	400	39
EXOR	10	3	30	37	50	74
3G	68	1	68	145	272	53
7G	17	1	17	85	136	62
TOTALS:	134		374	486	1012	48

POWER DISSIPATION - 2.71 WATTS

TOTAL PINS - 541, including 55 I/O PINS

INPUT CONNECTOR PINS - 40

OUTPUT CONNECTOR PINS - 15

*NOTE - 2-LEVEL METAL SYSTEM

AA - 5/8/72

DRA-3017 SUMMARY

Function	Total No.	Gate Complexity	Total Gate Complexity	Pins Used	Pins Available	Percent Used
FF	38	6	228	179	418	42
ANI	23	7	161	142	368	38
EXOR	11	3	33	44	55	80
3G	71	1	71	199	284	70
7G	4	1	4	22	32	68
TOTALS:	147		497	586	1157	50

POWER DISSIPATION - 3.52 WATTS

TOTAL PINS - 652, including 77 I/O PINS

INPUT CONNECTOR PINS - 42

OUTPUT CONNECTOR PINS - 35

AA - 5/8/72

DRA-3018 SUMMARY

Function	Total No.	Gate Complexity	Total Gate Complexity	Pins Used	Pins Available	Percent Used
FF	36	6	216	168	396	42
ANI	31	7	217	338	496	68
EXOR	9	3	27	38	45	84
3G	86	1	86	216	344	62
7G	16	1	16	87	128	67
TOTALS:	178		562	847	1409	60

POWER DISSIPATION - 3.97 WATTS

TOTAL PINS - 925, including 89 I/O PINS

INPUT CONNECTOR PINS - 55

OUTPUT CONNECTOR PINS - 34

AA - 5/8/72

RETURNED MATERIAL REPORT

I. DRA-3013:

A) SERIAL NO. 34533

1. BAD VIAS
2. NOT REPAIRED

B) SERIAL NO. 34540

1. FIRST TO SECOND METAL SHORT
2. REPAIRED AND RETURNED

C) SERIAL NO. 35022

1. UNKNOWN SHORTS
2. SHORTS BAKED OUT AND NOT RETURNED

D) SERIAL NO. 31306

1. FIRST TO SECOND METAL SHORT
2. REPAIRED AND RETURNED

II. DRA-3014:

A) SERIAL NO. 35021

1. FIRST TO SECOND METAL SHORT
2. REPAIRED AND RETURNED

B) SERIAL NO. 34007

1. BAD VIAS
2. NOT REPAIRED

C) SERIAL NO. 35208

1. BAD THIRD METAL AND/OR OXIDE STEPS
2. NOT REPAIRED

D) SERIAL NO. 35808

1. NO DEFECTS FOUND
2. POSSIBLE ARRAY TO P.C. BOARD CONNECTION
3. POSSIBLE A.C. SPEED PROBLEM
4. NOT RETURNED

cont'd ...

RETURNED MATERIAL REPORT

Page Two

DRA-3014 - cont'd

E) SERIAL NO. 34904

1. SECOND TO THIRD METAL SHORT
2. REPAIRED AND RETURNED

III. DRA-3015:

A) SERIAL NO. 33307

1. UNKNOWN SHORTS
2. SHORTS BAKED OUT AND RETURNED

IV. DRA-3016:

A) SERIAL NO. 33311

1. FIRST TO SECOND METAL SHORT
2. REPAIRED AND RETURNED

V. DRA-3018:

A) SERIAL NO. 34011

1. SECOND TO THIRD METAL SHORT
2. REPAIRED AND RETURNED

RELIABILITY

THE MOST RECENT RELIABILITY STUDY WAS PERFORMED BY TEXAS INSTRUMENTS INCORPORATED UNDER CONTRACT TO NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, GEORGE C. MARSHALL SPACE FLIGHT CENTER, MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812.

RESULTS OF THIS STUDY ARE CONTAINED IN REPORT NUMBER 03-71-27 (FINAL REPORT - PHASE II) "DEVELOPMENT OF QUALITY STANDARDS FOR BIPOLAR LSI DEVICES", APRIL 1971. CONTRACT NUMBER IS NAS8-21319, CONTROL NUMBER DCN 1-8-60-00152(1F) AND S1(1F) AND S2(1F).

APPENDIX III

ADDER OPERATIONS

The following tables summarize the adder arithmetic and logical operations that may be specified using TRANSLANG. The execution phase controls and the value determination for the ABT dynamic condition are indicated.

Notes:

1.	A Register Selection:	A	A1 | A2 | A3
		A0	All ZEROS
2.	B Register Selection:	B	Any B Register Select option
		$\overline{B}$	ONES complement (by TRANSLANG) of the specified B Register Select option
		0	All ZEROS
		1	All ONES
3.	Z Register Selection:	Z	CTR | LIT | AMPCR
		0	All ZEROS
4.	Inhibit 8 Bit Carry:	0	Allow carry into bytes
		1	Inhibit carry into bytes
5.	Adder Operation		As specified in Microprogramming Section

ARITHMETIC OPERATIONS

ADDER OPERATION	RESULT FORM	REGISTER SELECT A[1]	B[2]	Z[3]	IC8[4]	ADDOP[5]	ABT IS TRUE IF RESULT IS ALL
A ADD B		A	B	0	0	2	ONES
A ADD Z	R + S	A	0	Z	0	1	ONES
B ADD Z		0	B	Z	0	9	ONES
A ADL B		A	B	0	0	3	ZEROS
A ADL Z	R+S+1	A	0	Z	0	0	ZEROS
B ADL Z		0	B	Z	0	8	ZEROS
A CAD B	R + S	A	B	0	1	2	ONES
A CAD Z	WITH-OUT	A	0	Z	1	1	ONES
B CAD Z	CARRY	0	B	Z	1	9	ONES
0		0	0	0	0	2	NEVER
1		0	0	0	0	3	NEVER

MONADIC LOGICAL OPERATIONS

ADDER OPERATION	RESULT FORM	REGISTER SELECT A[1]	B[2]	Z[3]	ADDOP[5]	ABT IS TRUE IF RESULT IS ALL
A		A	0	0	2	ONES
B	R	0	B	0	2	ONES
Z		0	0	Z	1	ONES
NOT A		A	0	0	15	ZEROS
NOT B	$\bar{R}$	0	B	0	10	ZEROS
NOT Z		0	0	Z	12	ZEROS

DYADIC LOGICAL OPERATIONS

ADDER OPERATION	RESULT FORM	REGISTER SELECT A[1]	B[2]	Z[3]	ADDOP[5]	ABT IS TRUE IF RESULT IS ALL
A AND B		A	$\overline{B}$	0	7	ONES
A AND Z	R∧S	A	1	Z	13	ZEROS
B AND Z		0	$\overline{B}$	Z	4	ONES
A NIM B		A	B	0	7	ONES
A NIM Z	R∧$\overline{S}$		INVALID			
B NIM Z		0	$\overline{B}$	Z	13	ZEROS
A NRI B		A	$\overline{B}$	0	10	ZEROS
A NRI Z	$\overline{R}$∧S	A	0	Z	5	ONES
B NRI Z		0	B	Z	4	ONES
A NOR B		A	B	0	10	ZEROS
A NOR Z	$\overline{R}$∧$\overline{S}$		INVALID			
B NOR Z		0	B	Z	13	ZEROS
A XOR B		A	B	0	6	ONES
A XOR Z	(R∧$\overline{S}$)∨($\overline{R}$∧S)	A	0	Z	4	ONES
B XOR Z		0	$\overline{B}$	Z	14	ZEROS
A EQV B		A	$\overline{B}$	0	6	ONES
A EQV Z	(R∧S)∨($\overline{R}$∧$\overline{S}$)	A	0	Z	14	ZEROS
B EQV Z		0	B	Z	14	ZEROS
A NAN B		A	$\overline{B}$	0	15	ZEROS
A NAN Z	$\overline{R}$∨$\overline{S}$	A	1	Z	5	ONES
B NAN Z		0	$\overline{B}$	Z	12	ZEROS
A IMP B		A	B	0	15	ZEROS
A IMP Z	$\overline{R}$∨S		INVALID			
B IMP Z		0	$\overline{B}$	Z	5	ONES

DYADIC LOGICAL OPERATIONS (Cont'd)

ADDER OPERATION	RESULT FORM	REGISTER SELECT A[1]	B[2]	Z[3]	ADDOP[5]	ABT IS TRUE IF RESULT IS ALL
A OR B		A	B	0	11	ONES
A OR Z	R∨S		INVALID			
B OR Z		0	B	Z	5	ONES
A RIM B		A	$\overline{B}$	0	11	ONES
A RIM Z	R∨$\overline{S}$	A	0	Z	12	ZEROS
B RIM Z		0	B	Z	12	ZEROS

TRIADIC LOGICAL OPERATIONS

ADDER OPERATION		ADDOP[5]	RESULT	ABT IS TRUE IF RESULT IS ALL
TRY1	A, B, Z	4	$\overline{B}$ (A XOR Z)	ONES
TRY2	A, B, Z	5	$(\overline{A} \wedge Z) \vee (B \wedge \overline{Z})$	ONES
TRY3	A, B, Z	12	$A \vee B \vee \overline{Z}$	ZEROS
TRY4	A, B, Z	13	$(A \wedge Z) \vee (\overline{B} \wedge \overline{Z})$	ZEROS
TRY5	A, B, Z	14	(A ∨ B) EQV Z	ZEROS

APPENDIX IV

TRANSLANG SYNTAX

	Reference Page
<Program> ::= <Program Name Line><Body><End Line>	111
<Program Name Line> ::= PROGRAM <Program Name><Start Address>	111
<Program Name> ::= <Label>	111
<Label> ::= <Letter> | <Label><Letter> | <Label><Digit>	94
<Letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z	93
<Digit> ::= 0|1|2|3|4|5|6|7|8|9	93
<Start Address> ::= ADR <Hex Address> | <Empty>	111
<Hex Address> ::= <Hex Number>	111
<Hex Number> ::= <Hex Digit> | <Hex Number><Hex Digit>	111
<Hex Digit> ::= <Digit> | A | B | C | D | E | F	93
<Empty> ::= {The null string of characters}	93
<Body> ::= <Comment>|<Statement>|<Body> <Statement> | <Body><Comment>	111
<Comment> ::= COMMENT <Comment Words>;	111
<Statement> ::= <Label Part> <Line> <% Comment>	111

	Reference Page
<Label Part> ::= <Label> : | <Empty>	111
<Line> ::= <Label Constant>|<Start Address>|<Insert>|<Instruction>	111
<Label Constant> ::= <Label> * <Integer>	111
<Insert> ::= INSERT <Label> <Start Address>	112
<% Comment> ::= % <Comment Words> | <Empty>	112
<Comment Words> ::= <Comment Character> | <Comment Words> <Comment Character>	112
<Comment Character> ::= <Character> |·|#|&|$|[|]|||/	93
<Character> ::= <Letter>|<Digit>|<Single Space>|<Symbol>	93
<Single Space> ::= { One horizontal blank position }	93
<Symbol> ::= , | ; | + | - | : | = | % |(|) |*	93
<Instruction> ::= <Label Part> <Literal Assignment> | <Label Part> <N Instruction>	112
<Literal Assignment> ::= <Literal><Assignment Op> AMPCR | <Literal><Assignment Op> SAR | <Literal><Assignment Op> SAR; <Literal><Assignment Op> LIT | <Literal><Assignment Op> LIT ; <Literal><Assignment Op> SAR | <Literal><Assignment Op> LIT	94
<Literal> ::= <Integer> | COMP <Integer> | <Label> | <Label> -1	94
<Integer> ::= <Digit> | <Digit> <Integer>	94
<Assignment Op> ::= =: | =	93
<N Instruction> ::= <Unconditional Part> <Conditional Part>	95
<Unconditional Part> ::= <Component List>	95
<Component List> ::= <Component> | <Component List> ; <Component> | <Empty>	95
<Component> ::= <Ext Op> | <Logic Op> | <Successor>	95
<Ext Op> ::= <Mem Dev Op> | <Set Op> | <Mem Dev Op> , <Set Op> | <Set Op> , <Mem Dev Op> | <Empty>	100

	Reference Page
⟨Mem Dev Op⟩ ::= MR1 | MR2 | MW1 | MW2 | DL1 | DL2 | DU1 | DU2 | DR1 | DR2 | DW1 | DW2 | LDM | LDN	100
<Set Op> ::= SET <Cond Adjust Bit> | RESET GC	100
<Cond Adjust Bit> ::= INT | LC1 | LC2 | LC3 | GC1 | GC2	96
<Logic Op> ::= <Adder Op> <Inhibit Carry> <Shift Op> <Destination List>	103
<Adder Op> ::= 0 | 1 | <Monadic> | <Dyadic> | <Triadic> | <Empty>	103
<Monadic> ::= <Not> <A Select> | <Not> <B Select> | <Not> <Z Select>	103
<Not> ::= NOT | <Empty>	96
<A Select> ::= 0 | A1 | A2 | A3 | <Empty>	106
<B Select> ::= 0 | 1 | B | B <M> <C> <L> <Empty>	106
<M> ::= <Gating>	106
<C> ::= <Gating>	106
<L> ::= <Gating>	106
<Gating> ::= 0 | T | F | 1	106
<Z Select> ::= CTR | LIT | AMPCR | <Empty>	106
<Dyadic> ::= <Not> <A Select> <Binary Op> <Not> <B Select> | <Not> <B Select> <Binary Op> <Not> <Z Select> | <Not> <A Select> <Az Op> <Not> <Z Select>	103
<Binary Op> ::= NOR | OR | NIM | IMP | <Az Op>	103
<Az Op> ::= AND | XOR | EQV | NRI | RIM | NAN | ADD | + | ADL | CAD	103
<Triadic> ::= <Try Op> <A Select> , <B Select> , <Z Select>	103
<Try Op> ::= TRY1 | TRY2 | TRY3 | TRY4 | TRY5	103
<Inhibit Carries> ::= IC | <Empty>	103
<Shift Op> ::= R | L | C | <Empty>	103

	Reference Page
<Destination List> ::= <Asgn> <Dest> | <Destination List> <Asgn> <Dest> | <Asgn>	107
<Asgn> ::= , | =: | =	107
<Dest> ::= A1 | A2 | A3 | MIR | BR1 | BR2 | AMPCR | <Input B> | <Input Ctr> | <Input Mar> | <Input Sar>	107
<Input B> ::= B | BEX | BAD | BC4 | BC8 | BMI | BBE | BBA | BBI | BAI | BBAI | B4I | B8I	107
<Input Ctr> ::= CTR | LCTR | INC	107
<Input Mar> ::= MAR | MAR1 | MAR2 | LMAR	107
<Input Sar> ::= SAR | CSAR	107
<Successor> ::= WAIT | STEP | SKIP | SAVE | CALL | EXEC | JUMP | RETN	110
<Conditional Part> ::= <If Clause> <Cond Comp List> <Else Clause> | <If Clause> | <When Clause> <Cond Comp List> | <Empty>	95
<If Clause> ::= IF <Condition>	96
<Condition> ::= <Not> <Cond>	96
<Cond> ::= LST | MST | AOV | ABT | COV | SAI | RDC | EX1 | EX2 | HOV | <Cond Adjust Bit>	96
<Cond Comp List> ::= THEN <Component List>	95
<Else Clause > ::= ELSE <Successor> | <Empty>	96
<When Clause> ::= WHEN <Condition>	96
<End Line> ::= END	112

APPENDIX V

TRANSLANG RESERVED WORDS AND TERMINAL CHARACTERS

RESERVED WORDS

The following words are reserved in TRANSLANG and may not be used as labels.

		Reference Page
A	Zero (0) as A Select. Use <Empty>.	106
A0	Zero (0) as A Select. Use <Empty>.	106
A1	A1 Register A Select or destination operator.	106
A2	A2 Register A Select or destination operator.	106
A3	A3 Register A Select or destination operator.	106
ABT	Adder Bit Transmit dynamic condition from phase 3 of prior microinstruction doing Adder Op.	98, 99
ADD	Addition logic operator: X ADD Y = X+Y	104
ADL	Add + 1 logic operator: X ADL Y = X + Y + 1	104
ADR	Starting address for microsequence.	111
AMPCR	Alternate Microprogram Count Register Z Select into middle bytes of adder or destination operator from barrel switch 12 LS bits.	94, 106
AND	And logical operator: X AND Y = XY	104

		Reference Page
AOV	Adder overflow, dynamic condition of previous microinstruction using adder, true if addition results in overflow.	98, 99
B	B Register Input Select same as BTTT; or to B from barrel switch; destination operator.	106, 108
B4I	To B from adder "not 4 bit carry" OR MIR; destination operator	109
B8I	To B from adder "not 8 bit carry" OR MIR; destination operator.	109
BAD	To B from adder; destination operator.	108
BAI	To B from adder OR MIR; destination operator.	109
BBA	To B from adder OR barrel switch; destination operator.	109
BBAI	To B from adder OR BSW OR MIR; destination operator.	109
BBE	To B from external bus OR barrel switch; destination operator.	108
BBI	To B from prior MIR contents OR barrel switch; destination operator.	109
BC4	To B from adder "not 4 bit carry" replicated and shifted; destination operator.	108
BC8	To B from adder "not 8 bit carry" replicated and shifted; destination operator.	108
BEX	To B from external bus; destination operator.	108
BMI	To B from prior MIR contents; destination operator.	108
BR1	To Base Register 1 from barrel switch 2nd LS byte; destination operator.	107
BR2	To Base Register 2 from barrel switch 2nd LS byte; destination operator.	107
BSW	To B from barrel switch; destination operator	107
C	Circular shift right the entire adder output. Operation takes place in barrel switch.	105
CAD	Character add by carry inhibit between 8 bit characters (bytes). (Can use IC.) X CAD Y = X + Y IC	104

		Reference Page
LMAR	Literal register contents will be placed in MAR. Destination operator.	109
LST	Least significant bit of adder output, dynamic condition from phase 3 of previous micro-instruction doing adder op.	98
MAR	Memory address register destination operator: from barrel switch LS byte.	109
MAR1	Memory address 1 destination operator: same as BR1, MAR.	100
MAR2	Memory address 2 destination operator: same as BR2, MAR	100
MIR	Memory information register destination operator from barrel switch.	107, 108
MR1	Read from memory address BR1/MAR mem dev op.	100
MR2	Read from memory address BR2/MAR mem dev op.	100
MST	Most significant bit of adder output, dynamic condition from phase 3 of previous microinstruction doing adder op.	98
MW1	Write the content of MIR to memory address BR1/MAR mem dev op.	100
MW2	Write the content of MIR to memory address BR2/MAR mem dev op.	100
NAN	Not And logical operator: X NAN Y = $\overline{X} \vee \overline{Y}$	104
NIM	Not Imply logical operator: X NIM Y = $X\overline{Y}$	104
NOR	Nor logical operator: X NOR Y = $\overline{X}\,\overline{Y}$	104
NOT	Complement monadic or condition operator Not X = $\overline{X}$	96, 103
NRI	Not Reverse Imply logical operator: X NRI Y = $\overline{X} \vee Y$	104
OR	OR logical operator: X OR Y = X v Y	104
PROGRAM	Bracket word beginning a program.	111
R	Right shift end off the entire adder output, left fill with zeros. Operation takes place in barrel switch.	105

		Reference Page
RDC	Read complete bit: set when external data is ready for input to B, reset by testing.	97
RESET	Reset the Global condition bits. RESET GC.	102
RETN	Return: use AMPCR + 2 as address and as new content for MPCR. Successor.	110
RIM	Reverse Imply logical operator: X RIM Y = X v $\overline{Y}$	104
SAI	Switch Interlock accepts information bit. Set when switch interlock accepts information, reset by testing.	97
SAR	Shift Amount Register destination operator from LS bits of barrel switch or from literal assignment.	94
SAVE	Save the MPCR in AMPCR: use MPCR + 1 as micro-address and as next MPCR. Successor.	110
SET	Set the conditional bit specified: either LC1, LC2, LC3, INT, GC1 or GC2.	102
SKIP	Skip the next instruction; use MPCR + 2 as microaddress and as next MPCR. Successor.	110
STEP	Step to next instruction: use MPCR + 1 as micro-address and as next MPCR. Successor.	94, 110
T	True gating for B register.	106
THEN	Starts the true alternative of conditional instruction.	95
TRY1	Triadic Operator: TRY1 A, B, Z = $\overline{B}$ A $\overline{Z}$ v $\overline{B}$ $\overline{A}$ Z	103, 105
TRY2	Triadic Operator: TRY2 A, B, Z = A Z v B $\overline{Z}$	103, 105
TRY3	Triadic Operator: TRY3 A, B, Z = A v B v $\overline{Z}$	103, 105
TRY4	Triadic Operator: TRY4 A, B, Z, = A Z v $\overline{B}$ $\overline{Z}$	103, 105
TRY5	Triadic Operator: TRY5 A, B, Z, = Z A v Z B v $\overline{A}$ $\overline{B}$ $\overline{Z}$	103, 105
WAIT	Wait for condition microaddress is MPCR; MPCR and AMPCR unchanged. Successor.	110
WHEN	Starts a conditional instruction, has an implicit ELSE WAIT.	96
XOR	Exclusive Or logical operator: X XOR Y = X $\overline{Y}$ v $\overline{X}$ Y	104

TERMINAL CHARACTERS

		Reference Page
,	Assignment operator for destination operators.	107
;	Delimiter. Use is mandatory after a comment statement and between components in a statement.	94, 113
:	Terminator of label part of instruction or insert.	111
=:	Assignment operator for literal assignments or destination list.	107
+	Add operator.	103
-	Part of assignment in literal assignment statement.	94
*	Label constant separator for defines.	112
(	Prefix delimiter for redundant part of instruction.	93
)	Suffix delimiter for redundant part of instruction.	93
%%	Line terminator and in-line comment prefix.	113
=	Assignment operator for literal assignment or destination list.	107

APPENDIX VI

TRANSLANG ERROR MESSAGES

The first section of the Microtranslator parses the input file, a line at a time, and produces a listing of the file, N-instructions, and error messages. The error messages indicate that errors were made in the syntax or semantics of an instruction. They will be printed out in the following format giving the error number and the line number of the instruction as follows:

****ERROR NUMBER NNN IN LINE LLL****

where NNN is the error number and LLL is the sequence number of the instruction in the input file.

Error Number	Definition	
1	Label too large	(more than 15 characters)
2	CTR and MAR Conflict	(one receives BSW output; the other literal)
3	Duplicate MAR	(2 MAR destinations)
4	Duplicate B destination	
5	Missing comma	
6	Missing semicolon	

Error Number	Definition
7	Incorrect destination designator
8	Symbol undefined
9	Duplicate logical operator
10	Logic operator error
11	Colon equal comma or colon missing or misplaced
12	Duplicate Z select
13	Duplicate A select
14	Duplicate B select
15	B Gating error
16	Duplicate counter operations
17	More than one set operation
18	Reset error
19	Memory device error
20	Duplicate shift operation
21	Duplicate test condition
22	Duplicate successors
23	Successor error
24	Successor after ELSE error
25	Duplicate label
26	Literal used not in a literal assignment instruction (misspelled reserved word)
27	Condition error
28	Misplaced THEN
29	Misplaced ELSE
30	Misplaced integer
31	Integer too large
32	Too many quoted characters
33	Wrong register for receiving a literal
34	Undefined input mistaken for label, or misspelled reserved word
35	Address wanted for insert program less than current address, or misspelled reserved word
36	Reset not followed by proper identifier

Error Number	Definition
37	Set not followed by proper identifier
38	Undeclared label
39	Wrong type: minus sign used in a type one instruction
40	Stack operation removed, AMPCR goes directly to adder.
41	NOT error — "NOT" misused
61	Named insert program not on library
62	No END on file
63	Address error — present address > insert address

If a nanotable name is requested which has never been saved before, NO SUCH NANOTABLE is printed and a new name requested.

If a new nanotable is given a name already in use, DUPLICATE NANOTABLE NAME ERROR is printed and a new name is requested.

If labels have been used in a program without being declared, the following printout occurs upon conclusion of the listings.

```
LAST ADR  LABELS NOT FOUND

   2          STR
  3B          SERROR
   4          Y10
```

The address is the hexidecimal microprogram address of the last instruction using the label in a program.

APPENDIX VII

GLOSSARY

A Registers (A1, A2, A3): Each of the three A registers is functionally identical. The A registers are used for temporary data storage within the Logic Unit of the Interpreter and serve as a primary input to the adder.

Adder: The adder in the Logic Unit of the Interpreter, is a modified version of a straightforward carry lookahead adder. It is also used for executing logic operations.

Alternate Microprogram Count Register (AMPCR) The AMPCR is a 12-bit register in the Memory Control Unit of the Interpreter, which contains the jump or return address for program jumps and subroutine returns within a microprogram.

B Register: The B register is the primary input interface between the Logic Unit of the Interpreter and the Data/Program Memory or Devices (via the Switch Interlock). It also serves as the secondary input to the adder.

Barrel Switch: The barrel switch is a matrix of gates in the Logic Unit of the Interpreter, used to shift a parallel data word any number of places to the right or left in a single clock time.

Base Register 1 and 2 (BR1, BR2): The Base Registers are two 8 bit registers in the Memory Control Unit of the Interpreter, which usually contains the base address of a 256-word block of Data/Program Memory.

Building Block: The primary functional units of the Interpreter Based System: Interpreter, Data/Program Memory and the Switch Interlock.

Condition Register (COND): The COND is a 12-bit register in the Control Unit of the Interpreter and is used to store various condition bits for use during program execution.

Central Processor Unit (CPU): The primary arithmetic and control unit in a conventional computer system.

Condition Select: The condition select is a matrix of gates in the Control Unit of the Interpreter that compares the results of a computation or logical operation in the Logic Unit with a preselected result. The result of the comparison may be used to determine the sequence of execution of microprogram instructions.

Control Register (CR): The CR is a 38-bit register of the Interpreter which is used to store control signals from the Nanomemory that are not used in phase one of a clock cycle.

Control Unit (CU): The CU, one of the five major functional units of the Interpreter, is used for condition testing and the storage and distribution of enable signals received from the Nanomemory.

Counter (CTR): The CTR is an 8-bit counter in the Z register section of the Memory Control Unit of the Interpreter, used for loop control and other counting functions.

Data/Program Memory. The Data/Program Memory, also called S Memory, provides storage for data and program (either microprogram or conventional program in an emulation application) and functions similarly to the main memory modules of a conventional computer system.

Device: As used in the context of Interpreter-Based System, Devices include all the conventional computer system peripheral equipments such as disk files, magnetic tape units, high speed line printers, card readers, etc. and various sensors usually found in special data processing applications. The function of Devices is to provide the unique input/output medium for each system application.

Device Controller: A functional unit designed to interface and control a specific peripheral device (such as a disk file, magnetic tape unit, line printer, etc.) to the Input/Output module of a conventional computer system.

Device Dependent Port (DDP): The DDP permits any device to be interfaced with a Port Select Unit (PSU) by providing the specific device electrical interface such as logic level conversion, line driver/receiver capability, and timing and synchronization when required (as in the case of disk files, magnetic tape units, etc.)

Dual-In-Line (DIL): Describes the pin connection arrangement of one type of standard integrated circuit package.

End-Around Shift: A right shift operation in which the bit or bits which would be shifted out of the register are reinserted in the more significant end.

End-Off Shift: A shift operation in either the left or right direction, in which the bit or bits shifted out of the register are lost. Vacated bit positions may be automatically filled with zeros.

Firmware: In the Interpreter-Based System, firmware is the combination of stored logic in the M and N memories of the Interpreter.

Incrementer (INCR): The INCR is in the Memory Control Unit of the Interpreter and increments by zero, one, or two, the address of the next microinstruction to be executed by the Interpreter.

Input/Output Module (I/O): The I/O is the interface and control unit between the CPU and peripheral input/output devices in a conventional computer system.

Interpreter: The Interpreter is the basic building block of the Interpreter-Based System. Functionally, it is characterized by the combination of macroprogram instructions stored in its M memory and hardware logic enabled by a multiplicity of enable signals stored in its N memory.

Interpreter-Based System: A computer organization and implementation concept that provides, in configurations of basic building blocks, the throughput and flexibility for a variety of data processing requirements.

Large Scale Integration (LSI): The implementation of more than 100 bipolar logical gates in a single integrated circuit chip.

Least Significant Bit (LSB): For a number or value represented in binary notation, that bit position which represents the least significant portion of the number.

Literal Register (LIT): An 8-bit register in the Z register section of the Memory Control Unit of the Interpreter, which is used for temporary storage of literals from microinstructions.

Logic Unit (LU): The LU, one of the five major functional units of the Interpreter, performs all of the arithmetic, Boolean logic, and shifting operations of the Interpreter.

Medium Scale Integration (MSI): The implementation of 20 to 100 bipolar logical gates in a single integrated circuit chip.

Memory Address Register (MAR): The MAR is an 8-bit register in the Memory Control Unit of the Interpreter, which contains the least significant 8 bits of a memory or device address.

Memory Control Unit (MCU): The MCU, one of the five major functional units of the Interpreter, controls the sequence of execution of microinstructions, the addressing of Data/Program Memory, and the selection of Devices.

Memory Information Register (MIR): The MIR is a register in the Logic Unit of the Interpreter which serves as the output interface register between the Interpreter and the Switch Interlock.

Microinstruction: A single instruction stored in M memory of the Interpreter, sequences of which characterize the Interpreter for a given microprogram. A microinstruction may contain an N memory address or a literal.

Microprogram Address Control Register (MPAD CNTL): The MPAD CNTL, a register in the Memory Control Unit of the Interpreter, controls the loading of the MPCR, the AMPCR, and controls the value of the increment.

Microprogram Address Section (MPAD): The MPAD is a collection of registers and controls in the Memory Control Unit of the Interpreter, which addresses the M memory for the sequencing of microinstructions.

Microprogram Count Register (MPCR): The MPCR, located in the Memory Control Unit of the Interpreter, is a 12-bit register that usually contains the address, in M memory, of the microinstruction currently being executed by the Interpreter.

Microprogram Memory (M Memory): The M memory, one of the five major functional units of the Interpreter, stores microinstructions which characterize the Interpreter for a given application, and may be implemented as a read/write semiconductor memory.

Microprogram Memory Buffer (MPB): The MPB buffers blocks of microinstructions read from a microprogram source in order to maintain the clock period of the Interpreter.

Most Significant Bit (MSB): For a number or value represented in binary notation, that bit position which represents the most significant portion of the number, or the sign of the number.

Multiprocessor: A network of computers capable of simultaneously executing two or more programs or sequences of instructions by means of multiprogramming, parallel processing, or both.

Nanoinstruction: A single instruction stored in N memory of the Interpreter, the contents of which constitute 56 unique signals for controlling the hardware logic of the Interpreter.

Nanomemory (N Memory): The N memory, one of the five major functional units of the Interpreter, stores 56 specific enable signals for the hardware logic within the Logic Unit, Control Unit, and Memory Control Unit.

Random-Access-Memory (RAM): A memory in which the time to access data is independent of its location in the memory, or of the data most recently accessed in the memory. By convention, a read/write memory.

Port Select Unit (PSU) The PSU provides control and the electrical interface between a single Interpreter and Devices and Data/Program Memory. The PSU is used in lieu of the Switch Interlock in system configurations that require only one Interpreter.

Random-Access-Memory (RAM): A memory in which the time to access data is independent of its location in the memory, or of the data most recently accessed in the memory.

Read-Only Memory (ROM): A memory that stores data not alterable by program instruction.

Remote/Card: A program subroutine executed on a Burroughs B 5500 which permits a user to create card images of TRANSLANG instructions on a disk file, using a remote terminal of the B 5500.

Shift Amount Register (SAR) The SAR is a 6-bit register in the Control Unit of the Interpreter and is used to store the number of positions a word or literal is to be shifted by the barrel switch.

Small Scale Integration (SSI) The implementation of 5 to 20 logical gates in a single integrated circuit chip.

Switch Interlock (SWI): The SWI provides the interconnection between Interpreters, Data/Program Memory, and Devices of an Interpreter-Based System. Its function is to permit any one of a multiplicity of Interpreters to access all modules of an array of Data/Program Memory and/or all Devices.

Transistor-Transistor-Logic (TTL): A family of transistor circuits used to implement digital logic networks, and characterized by its high speed, large capacitance drive capability and excellent noise immunity.

TRANSLANG: A computer program designed to convert English language statements defining the action of the Interpreter for each machine clock cycle, into binary patterns for the M and N memories.

Z Register Section: A collection of registers and selection gates in the Memory Control Unit of the Interpreter, which include the CTR, LIT, and Input Selection gates used to control the execution sequence of microinstructions.

REFERENCES

1. O. L. MacSorley, "High Speed Arithmetic in Binary Computers" Proceedings of the IRE (January 1963) pp. 67-91.

2. W. A. Curtin, "Multiple Computer Systems" Advances in Computers, Vol. 4 (1963) Ed: F. L. Alt and M. Rubinoff, New York Academic Press, 1963.

3. R. C. Larkin, "A Minicomputer Multiprocessing System" Proceedings of Computer Designers Conference; Anaheim, California (January 1971) pp. 231-235.

4. Hughes Aircraft Co., "Seek Flex Preliminary Design Study, Volume 1: System Design and Rationale" Ground System Group Report FR71-16-430 (July 23, 1971).

5. J. D. Meng, "A Serial Input/Output Scheme for Small Computers" Computer Design Vol. 9, No. 3 (March 1970) pp. 71-75.

6. R. G. Buus, "Electrical Interference" Physical Design of Electronic Systems Vol. I, pp. 416-434, Prentice Hall, 1970.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Advanced Development Organization
Burroughs Corporation
Defense, Space and Special Systems Group

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

AEROSPACE MULTIPROCESSOR FINAL REPORT

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report Covers Period June 1970 through May 1, 1973

5. AUTHOR(S) (First name, middle initial, last name)
Robert L. Davis
Sandra Zucker

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1973	208	6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F33615-70-C-1773	64161
b. PROJECT NO. 6090	
c. Task 01	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	AFAL-TR-73-114

10. DISTRIBUTION STATEMENT Distribution limited to U. S. Government agencies only; test and evaluation results reported March 1973. Other requests for this document must be referred to Air Force Avionics Laboratory (AAM), Wright-Patterson Air Force Base, Ohio 45433

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
Submitted by the author to Air Force Avionics Laboratory in March 1973	Air Force Avionics Laboratory (AFSC) Info. Mgt. Branch, Sys. Avionics Div. Wright-Patterson AFB, Ohio

13. ABSTRACT

The aerospace multiprocessor described is based upon a modular, building block approach. An exchange concept that is expandable with the number of processors, memory modules, and device ports, was developed whose path width is a function of the amount of serialization desired in the transmission of data and address through the exchange. The processors (called Interpreters) are microprogrammable utilizing a 2-level microprogram memory structure and were designed for implementation with large scale integrated circuits. The modularity exhibited in the Interpreters is in the size of the microprogram memories and in the word length of the Interpreters from 8 bits through 64 bits in 8-bit increments.

The specific implementation of the exchange for the aerospace multiprocessor is for five processors, eight memory modules, and eight device ports with eight wires each carrying four serial bits of data through the exchange. The processors each have word lengths of 32 bits with a 512 word × 15 bits first-level microprogram memory and a 256 word × 54 bit second-level microprogram memory.

A simplified control program based upon concepts for a modular executive structure, and some user type programs were written for demonstration of the aerospace multiprocessor.

DD FORM 1 NOV 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Aerospace Multiprocessor						
Interpreter						
Microprogramming						
Multiprocessor						
Switching Interlock						